Hecht

DAS GROSSE COMMODORE 64 BUCH

Das Buch, das keine Fragen offen läßt: Programmierung von Grafik, Sound und Floppy, in BASIC und Assembler. Anwendungen wie GEOS 2.0, DFÜ, DTP, Textverarbeitung, Spiele...

DATA BECKER

Hecht

DAS GROSSE COMMODORE 64 BUCH

DATA BECKER

Copyright © 1989 by DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf 1

2. unveränderte Auflage 1989

Umschlaggestaltung Harald Müller

Text verarbeitet mit Word 4.0, Microsoft

Druck und buchbinderische Verarbeitung Elsnerdruck, Berlin

ISBN 3-89011-370-2

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle technischen Angaben und Programme in diesem Buch wurden von dem Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Aus drucktechnischen Gründen sind bei den folgenden Listings einige Zeilen in eine neue Zeile umgebrochen worden, z.B.:

```
270 A = AD: GOSUB 3240: PR$ = A$ + "< 2 SPACE>": GOSUB 2000: IF LEFT$ (X$,3)
= ".EN" THEN 1000
```

Geben Sie diese bitte als eine Zeile ein.

Listing:	ASSEMBLER	Seite 414 bis 422
	SIMULATOR	Seite 424 bis 432
	DISASSEMBLER	Seite 432 bis 435

Vorwort

Weit über eine Million Commodore 64 wurden bisher allein in Deutschland verkauft und trotz starker Konkurrenz - durch den Amiga 500 auch aus eigenem Hause - hält sich der gute, alte 64'er nach wie vor erstaunlich gut. Das ist auch nicht weiter verwunderlich. Nicht zuletzt dank des mittlerweile extrem günstigen Preises (als ich mir vor fast vier Jahren meinen 64'er zulegte, kostete er noch rund 600 Mark) dürfte der Commodore 64 nach wie vor der ideale Einsteigercomputer sein. Was man allerdings schmerzlich vermißt, ist eine ausführliche Dokumentation. Das spärliche Handbuch wird wohl jeder bald enttäuscht zur Seite legen.

An diesem Punkt setzt das große Buch zum Commodore 64 an. Ich möchte Ihnen in diesem Buch zeigen, was alles im Commodore 64 steckt und - vor allem - wie man ihn sich nutzbar macht. Allein zum Spielen ist der 64'er nämlich viel zu schade! Ein großer Schwerpunkt bildet dabei die vielleicht kreativste Tätigkeit im Zusammenhang mit Computern: das Programmieren. Gerade beim Commodore 64 ist es ein Leichtes, sich die Grundlagen der Programmierung zu erarbeiten. Dabei möchte ich Sie durch zwei große Einführungskurse in die Sprachen BASIC und Assembler und viele praktische Tips und Hilfen zur Systemprogrammierung tatkräftig unterstützen.

Vielleicht lesen Sie dieses Vorwort gerade in einem Computergeschäft und fragen sich nun, ob es sich für Sie lohnt, das Buch zu kaufen. Und wenn Sie es sich schon gekauft haben, dann kann ein kleiner Überblick ja erst recht nicht schaden. Schauen wir uns also einmal im Detail an, was Sie in den einzelnen Kapiteln an Themen erwartet.

Kapitel 1 enthält eine Einführung in den Commodore 64. Falls Sie sich Ihren 64'er erst vor kurzem gekauft haben und noch nicht so recht wissen, wo Sie was anschließen müssen oder wie man die verschiedenen Arten von Software zum Laufen bringt, so steht Ihnen Kapitel 1 mit Rat und Tat zur Seite.

In Kapitel 2 möchte ich Ihnen zeigen, was man auf dem Commodore 64 mit Hilfe kommerzieller Hardware- und Software-Produkte alles machen kann. Neben den "klassischen" Anwendungsbereichen, wie etwa Text- und Datenverarbeitung, geht es dabei auch um so aktuelle Themen wie, Desktop Publishing oder Daten-Fernübertragung (DFÜ). Und natürlich wurde auch der Spielebereich nicht vergessen. Einen weiteren Schwerpunkt bildet der Bereich "Lern-Software". Der Computer ist schließlich immer noch der geduldigste Lehrer und im Endeffekt auch wesentlich billiger als Nachhilfestunden.

In Kapitel 3 steigen wir dann in die Programmierung ein. In einem ausführlichen und leicht verständlichen Kurs erfahren Sie alles über die Programmiersprache BASIC. Schon nach wenigen Tagen werden Sie in der Lage sein, Ihre ersten eigenen Programme zu schreiben.

Nachdem Sie sich in BASIC eingearbeitet haben, wird es Ihnen auch nicht mehr schwer fallen, die zweite Programmiersprache des Commodore 64, Assembler oder Maschinensprache (mit beidem meint man letztendlich dasselbe), zu erlernen. Maschinensprache wird gerne als "Profi-Sprache" bezeichnet, da sie erheblich schneller und effizienter, dafür aber auch schwerer zu handhaben ist als BASIC. Daß der letzte Punkt nicht unbedingt zutreffen muß, zeigt Kapitel 4. Hier erhalten Sie den letzten "Schliff" auf dem Weg zum Programmier-Profi.

Die Kapitel 5 bis 8 befassen sich mit der Systemprogrammierung. Wie Sie schon bald feststellen werden, unterstützt die Programmiersprache BASIC die Fähigkeiten des Commdore 64 nur sehr unzureichend. Ich werde Ihnen deshalb eine Vielzahl "gebrauchsfertiger" Maschinenprogramme vorstellen, mit denen Sie das BASIC ihres 64'er "tunen" können. Alle derartigen Programme können Sie dank sogenannter BASIC-Lader auch ohne jegliche Maschinensprache-Kenntnisse nutzen. In praktisch allen Ihren Programmen werden Sie irgendwelche Daten dauerhaft speichern müssen. Die Datenverwaltung mit Floppy und Datasette wird daher in den Kapiteln 5 und 10 ausführlich besprochen.

Kapitel 6 widmet sich dem vielleicht faszinierendsten Programmiergebiet auf dem Commodore 64: der Grafikprogrammierung. Hier erfahren Sie alles Wissenswerte zur Programmierung der hochauflösenden Grafik, zur Arbeit mit den sogenannten Sprites und zum Entwurf eigener Zeichensätze.

Kapitel 7 hat ein nicht minder faszinierendes Thema zum Gegenstand: die Programmierung von Sound und Musik.

Kapitel 8 befaßt sich mit den etwas spezielleren Schnittstellen des Commodore 64, den Joystick-Anschlüssen, dem User-Port und dem Expansion-Port. Besonders interessant dürfte dabei die Realisierung einer RS-232-Schnittstelle am User-Port sein.

Technisch Interessierte erfahren in Kapitel 9 alles Wissenswerte zur Wartung und Pflege ihrer Floppy 1541. Kapitel 10 hingegen ist eine wahre Fundgrube für alle Besitzer der Datasette 1530. Daß man als Nicht-Floppy-Besitzer durchaus auch mit der Datasette vernünftig arbeiten kann, wird hier gezeigt. Sogar ein neues Betriebssystem, das unter anderem bis zu 20-mal schnelleres Laden erlaubt, ist Bestandteil dieses Kapitels.

Der Anhang schließlich enthält eine umfangreiche Sammlung nützlicher Referenzen und Tabellen zum schnellen Nachschlagen. Sie sehen also, es erwartet Sie eine riesige Fülle an hochinteressanten und überaus nützlichen Informationen. Bleibt mir nur, Ihnen viel Spaß beim Lesen und viel Erfolg bei der Arbeit mit Ihrem Commodore 64 zu wünschen.

Doch ich möchte nicht vergessen, all den Leuten herzlich zu danken, die mir bei der Fertigstellung dieses Buches behilflich waren und mich insbesondere für das Kapitel 2 mit vielen Tips und Informationen versorgt haben. Mein ganz besonderer Dank gilt jedoch Rolf Brückmann, Lothar Englisch, Jaques Felt, Ralf Gelfand, Klaus Gerits, Reinhold Herrmann, Rüdiger Kerkloh, Darko Krsnik, Hans Joachim Liesert, Dirk Paulissen, Andreas Polk, Michael Strauch sowie Manfred und Helmut Tornsdorf.

Martin Hecht *Stuttgart, im Oktober 1989*

Inhaltsverzeichnis

1. Einführung

Sie haben sich also einen Commodore 64 gekauft. Und nun brennen Sie natürlich darauf, Ihrem Gerät die ersten Reaktionen zu entlocken. Doch schön der Reihe nach. Bevor wir uns den 64'er genauer anschauen, möchte ich Sie zunächst mit den beiden vielleicht wichtigsten Fachbegriffen in der Welt der Computer vertraut machen: der Hardware und der Software.

All das, was man beim Computer anfassen kann, bezeichnet man allgemein als Hardware. Dazu zählen neben der Computerkonsole selbst beispielsweise die Anschlußkabel, das Netzteil, aber auch Bildschirme, Drucker und sonstige Zusatzgeräte. Vereinfacht gesagt handelt es sich bei der Hardware also um elektrische und elektronische Bauteile im weitesten Sinne.

Mit der Hardware allein läßt sich noch nichts anfangen. Um den Computer zum Leben zu erwecken, benötigt man Software. Damit meint man die Programme, die der Hardware "sagen", was sie tun soll. Solche Programme können Sie auch selbst schreiben. Der Commodore 64 verfügt dazu über eine eingebaute Programmiersprache, das BASIC. Dazu jedoch später mehr.

Vielleicht wundern Sie sich, warum ich Ihnen all das gleich ganz am Anfang erkläre. Das hat jedoch einen wichtigen Grund: Während man bei der Arbeit mit Software grundsätzlich am Rechner nichts "kaputt machen" kann, sollte man beim Umgang mit der Hardware sehr vorsichtig sein. Ein falsch angeschlossenes Kabel oder ein unbedachter Griff an ein elektronisches Bauteil kann sehr schnell ernste und vor allem kostspielige Schäden am Computer nach sich ziehen. Wenn Sie dagegen Software nutzen, also zum Beispiel ein BASIC-Programm schreiben oder mit einer Textverarbeitung einen Brief erstellen, bekommen Sie im schlimmsten Fall einen sogenannten "Systemabsturz", nach dem der Rechner zu keiner vernünftigen Reaktion mehr zu bewegen ist. Solch ein Systemabsturz läßt sich leicht beheben: einfach den Rechner aus- und wieder einschalten. Danach ist alles wieder beim alten! Irgendein Schaden am Rechner entsteht nicht.

Ganz ohne Folgen ist allerdings auch ein solcher Absturz, der in der Regel durch einen Fehler im abgearbeiteten Programm verursacht wird, nicht. Durch das Ausschalten des Rechners wird nämlich der Speicher des Commodore 64 vollständig gelöscht. Das Programm und die Daten, die sich evtl. im Rechner befanden, gehen dadurch unwiderbringlich verloren. Aber auch dieses Problem läßt sich leicht lösen, indem man seine Programme und Daten auf einem dauerhaften Speichermedium abspeichert. Dazu gleich mehr.

1.1 Den C64 kennenlernen

Nachdem Sie den Commodore 64 ausgepackt haben, müßten Sie drei "Einzelteile" vor sich liegen haben. Die Computerkonsole, ein Netzteil sowie ein Kabel, mit dem Sie den Rechner mit einem Fernseher verbinden können.

Der im Netzteil eingebaute Trafo versorgt den Commodore 64 mit der erforderlichen Versorgungsspannung. Nach einigen Stunden Betrieb wird das Netzteil recht heiß und eignet sich dann vorzüglich als Fußwärmer. Ansonsten ist an diesem Bauteil aber nichts weiter interessant.

Schauen wir uns also die Computerkonsole einmal genauer an. Im Gegensatz zu professionellen Personalcomputern, die meist über eine abgesetzte Tastatur verfügen, hat man beim Commodore 64 alles in einem Stück. Die Tastatur erinnert an eine Schreibmaschinentastatur. Wenn Sie allerdings genauer hinschauen, werden Sie feststellen, daß die deutschen Umlaute fehlen und einige Tasten, insbesondere <Z> und <Y>, anders angeordnet sind. Der 64'er kann seine Herkunft nicht verleugnen. Er verfügt über eine amerikanische Tastaturbelegung. Ich werde Ihnen aber in Kapitel 6, in dem wir uns der Grafikprogrammierung widmen, zeigen, wie man ihm, wenn auch auf Umwegen, doch noch zu einer deutschen Belegung verhelfen kann.

Auf der rechten Seite sowie auf der Rückseite des Gehäuses befinden sich allerlei Anschlüsse. Für sie im Moment am wichtigsten sind zwei Anschlüsse: die Spannungsversorgung und der

TV-Anschluß. Dabei möchte ich einmal davon ausgehen, daß Sie den Commodore 64 an einen Fernseher anschließen wollen.

Auf der rechten Seite des Gehäuses ganz hinten finden Sie den Anschluß für die Spannungsversorgung. Hier müssen Sie den Gerätestecker vom Netzteil einstecken. Der andere Stecker des Kabels kommt in die Steckdose. Das dürfte klar sein.

Direkt neben dem Spannungsversorgungsanschluß befindet sich der Ein-/Ausschalter. Sobald Sie den Schalter umlegen, leuchtet die rote LED an der Oberseite des Gehäuses und zeigt Ihnen an, daß der Commodore 64 betriebsbereit ist. Einschalten sollten Sie den Rechner allerdings erst, sobald Sie ihn mit dem Fernseher verbunden haben. Also noch etwas Geduld.

Daß sich der Ein-/Ausschalter an der Computerkonsole und nicht am Netzteil selbst befindet, hat einen kleinen Nachteil. Dadurch wird das Netzteil nämlich auch nach dem Ausschalten des Rechners nicht völlig vom Stromnetz abgetrennt. Bemerkbar macht sich das an einer leichten Erwärmung des Netzteils, auch wenn Sie den Commodore 64 längere Zeit gar nicht eingeschaltet hatten. Für das Netzteil ist das aber nicht weiter schädlich. Trotzdem empfiehlt es sich, zusätzlich den Netzstecker zu ziehen, wenn Sie längere Zeit nicht mit dem Rechner arbeiten.

Die beiden Anschlüsse vor dem Ein-/Ausschalter sind für sogenannte Joysticks gedacht. Diese sind insbesondere zur Steuerung von Spielen sehr wichtig. Mit einem Joystick können Sie zum Beispiel ein Raumschiff oder sonst eine spielerische Figur über den Bildschirm steuern. Aber auch die Benutzeroberfläche GEOS läßt sich mit einem Joystick bedienen. Für manche Anwendungen ist dem Joystick eine sogenannte Proportional-Maus vorzuziehen. Diese kann ebenfalls an einem der beiden "Control-Ports", so der Name der Anschlüsse, angeschlossen werden.

Daß es gleich zwei Control-Ports gibt, ist nicht ohne Grund: Viele Spiele bieten nämlich die Möglichkeit, zu zweit zu spielen. In diesem Fall steuert dann der eine Spieler seine Spielfiguren über Control-Port 1 und der zweite Spieler seine über Control-Port 2. Nicht wenige Programme unterstützen aber nur einen der

beiden Anschlüsse. Die Frage ist dann, welchen? Da kann es Ihnen leicht passieren, daß Sie den Joystick am Control-Port 1 anschließen, das Programm unterstützt aber nur den Control-Port 2. Ein Blick in die Bedienungsanleitung des betreffenden Programms sorgt hier meist schnell für Klarheit.

Auf der Rückseite des Gehäuses befinden sich die meisten Anschlüsse. Fangen wir ganz links an. Da wäre zunächst der sogenannte User-Port. Dabei handelt es sich um einen etwas speziellen Anschluß, der sich aber sehr flexibel einsetzen läßt. So läßt sich hier zum Beispiel ein Drucker aus dem PC-Bereich anschließen.

Daneben finden Sie den Anschluß für die Datasette. Das ist ein Kassettenlaufwerk, mit dem Sie auf handelsüblichen Musikkassetten Programme und Daten aufzeichnen können. Die Datasette bezieht auch ihre Stromversorgung aus diesem Anschluß, so daß Sie keinen separaten Stromanschluß benötigen.

Der runde Anschluß daneben ist zum Anschluß einer Floppy, einem Diskettenlaufwerk, gedacht. Eine Floppy hat im Vergleich zu einer Datasette viele Vorteile, ist aber auch wesentlich teuerer. Ein von Commodore angebotener Drucker kann ebenfalls hier angeschlossen werden. Wenn Sie mit beidem arbeiten wollen, also Floppy und Drucker, dann schließen Sie die Floppy am Commodore 64 an und den Drucker anschließend an der Floppy. Die Floppy verfügt dazu an der Rückseite über zwei entsprechende Anschlüsse.

Der nächste, ebenfalls runde Anschluß dient zum Abnehmen des Audio- und des Videosignals. Wenn Sie über einen passenden Monitor verfügen, können Sie diesen hier direkt anschließen. Zudem ist es möglich, das Tonsignal isoliert abzunehmen, um es beispielsweise einer Stereoanlage zuzuführen. Ein entsprechendes Kabel ist im einschlägigen Fachhandel erhältlich.

Nun kommen wir zum für Sie im Moment wohl wichtigsten Anschluß: dem TV-Anschluß, der sich direkt neben dem Audio- und Videoausgang befindet.

Doch haken wir, um Ihre Neugier zu befriedigen, noch schnell den letzten Anschluß ab. Der große Schacht, der sich neben dem Fernsehanschluß befindet, ist ein sogenannter Modul-Steckplatz. Die Auslieferung von Software auf Modulen ist heutzutage nicht mehr sehr weit verbreitet; die meiste Software wird heute auf Diskette geliefert. Trotzdem hat der Modul-Steckplatz durchaus seine Berechtigung. Mehr dazu etwas weiter unten, wenn wir uns mit dem Umgang mit Software beschäftigen.

Zurück zum TV-Anschluß. Ihrem Commodore 64 liegt ein Kabel bei, mit dem Sie den Rechner mit einem Fernseher verbinden können. Dazu stecken Sie den Flügelstecker mit langem Stift in den TV-Anschluß des Rechners. Der andere Stecker kommt in den Antennenanschluß des Fernsehers.

Das ist schon fast alles. Sie müssen jetzt nur noch Ihr Fernsehgerät passend einstellen. Dazu müssen Sie wissen, daß sich der Commodore 64 praktisch wie ein zusätzliches Fernsehprogramm verhält. Und zwar "sendet" Ihr Commodore 64 auf Kanal 36. Suchen Sie sich also eine freie Stationstaste, und stellen Sie sie auf diesen Kanal ein. Wenn Sie jetzt nicht genau wissen, wie das bei Ihrem Fernsehgerät geht, müssen Sie nicht extra nach der Bedienungsanleitung kramen. Schalten Sie einfach beide Geräte, Commodore 64 und Fernsehgerät, ein, und schalten oder schrauben Sie so lange an den Schaltern und/oder Schrauben der Stationstaste, bis sich ein gut lesbares Einschaltbild ergibt.

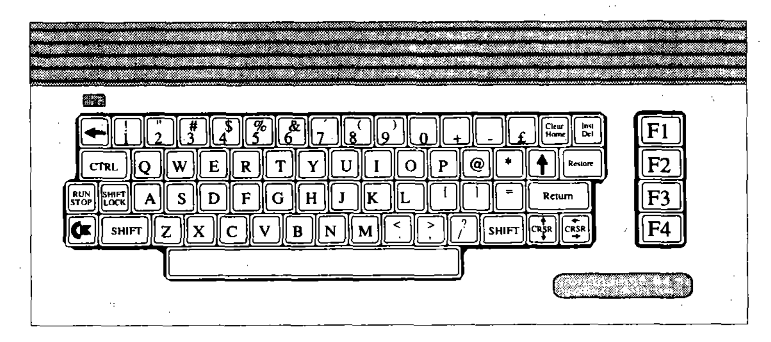
Wenn Sie alles richtig gemacht haben, müßte auf Ihrem Fernseher nun folgendes Bild zu sehen sein:

```
     **** COMMODORE 64 BASIC V2 ****
  64K RAM SYSTEM  38911 BASIC BYTES FREE

READY.
■
```

Der Text erscheint dabei in hellblauer Schrift auf dunkelblauem Grund. Der Rahmen hat ebenfalls eine hellblaue Farbe.

Die Tastatur



Das wichtigste Eingabegerät bei Ihrer zukünftigen Arbeit mit dem Commodore 64 wird die Tastatur sein. Daher lernen Sie in diesem Kapitel die Handhabung der 66 Tasten Ihres C64. Mit Hilfe dieser Tasten werden Sie alle verborgenen Möglichkeiten Ihres Heimcomputers erschließen. Sei es die Eingabe von Texten, das Erstellen von Bildern (Grafiken), das Abschicken von Anweisungen oder die Berechnung mathematischer Probleme, all dies wird für Sie nach diesem Kapitel kein Neuland mehr sein. Wenn Sie einmal "alle" Tasten beherrschen, wird der C64 Ihnen ein folgsamer Partner bei der Lösung vieler Probleme sein.

Das Beherrschen der Tastatur dient nicht nur Computer-Freaks, die einmal jedes Bit des Rechners mit dem Vornamen kennen möchten, sondern auch denen, die ihren Computer nur mit den reichhaltig angebotenen Fertigprogrammen "füttern" wollen. Jemand, der sich zu den zuletzt genannten Anwendern zählt, darf bei einer Meldung eines Standardprogramms, wie z.B. "Drücken Sie die <Clr/Home>-Taste zum Ausdruck des Bildschirminhalts", nicht verzweifelt in seinem Handbuch wühlen.

Kurzum sollte jeder, der auch nur gelegentlich an dem C64 arbeitet, mit der Tastatur vertraut sein. Der Vater eines computerfaszinierten Sohnes z.B. sollte zumindest wissen, wie das beste Spielprogramm des Sohnemanns geladen wird, wenn der gerade Fußball spielt.

Doch keine Angst, Sie müssen sich nicht vorher zu einem Schreibmaschinenkursus an der Volkshochschule anmelden. Die

meisten auch noch so erfahrenen Freizeitprogrammierer arbeiten mit dem "Zweifinger-Suchsystem". Wer eine gewisse Zeit mit der Tastatur vertraut ist, wird sich bald darüber wundern, wie flink er über die Tasten saust. Auch die oben beschriebenen Hilfstasten werden besonders ausführlich beschrieben. Diese Tasten sind sehr wichtig, da damit Farben bestimmt, Texte eingegeben, Programme unterbrochen und Befehle an den Rechner übermittelt werden können.

Allgemeines zur Tastatur

Auf den ersten Blick erweckt die Tastatur des C64 den Eindruck einer üblichen Schreibmaschinentastatur. Doch wenn Sie genauer hinsehen, werden Sie leichte Abweichungen feststellen:

- Die Buchstaben "Y" und "Z" sind gemäß der amerikanischen ASCII-Norm vertauscht.
- Die Tastatur weist keine Umlaute (ö, ä, ü) und kein scharfes "s" (ß) auf, da diese Zeichen im amerikanischen Zeichensatz nicht enthalten sind.
- Es gibt zusätzliche Tasten (Hilfstasten), deren Funktion später erläutert wird.

Sie sollten keine Experimente mit der Tastatur machen, bevor Sie nicht mit deren Funktion vertraut sind. Zwar können durch Fehlbedienungen keine Rauchwolken aus dem Rechner aufsteigen, doch Sie können durch überraschende Mißerfolge verblüfft werden. Warten Sie also geduldig auf die praktische Einweisung auf den nächsten Seiten.

Los geht's

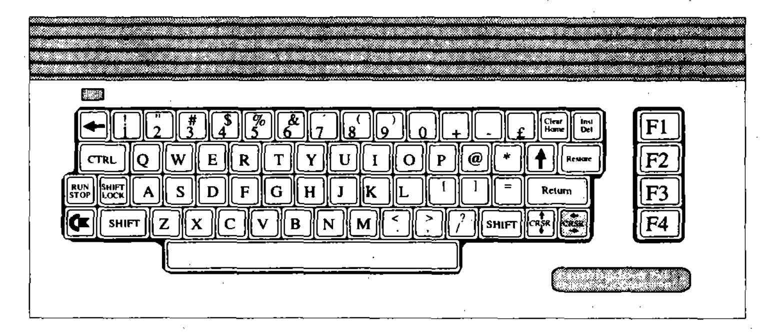
Nun wird es Zeit, die schlafende Technik Ihres C64 zum Leben zu erwecken. Schalten Sie dazu den Rechner ein. Nach jedem Einschalten erscheint eine Meldung, die besagt, daß der Rechner nun bereit ist, Ihren Kommandos zu folgen. Die erste Zeile der Meldung weist auf die interne BASIC-Version hin. Der C64 ist mit dem Commodore BASIC Version 2 ausgerüstet. Darüber hinaus gibt es noch eine Version 4, die aber nur in den größeren Commodore-Rechnern zu finden ist.

Die zweite Zeile der Einschaltmeldung informiert Sie darüber, daß ein 64 KByte-Speicher vorhanden ist (65535 speicherbare Zeichen). Von diesen 64 KByte stehen für Ihre persönliche Speicherung 38911 Bytes (Zeichen) zur Verfügung. Das ist eine ganze Menge, die von einem Programm allein meist nicht ausgenutzt wird. Oft wird diese enorme Speicherkapazität zur rechnerinternen Datenspeicherung benutzt. D.h., die vom Programm zu verarbeitenden Daten befinden sich komplett im Speicher.

In der folgenden Zeile besagt die Meldung READY, daß das Betriebssystem nun Kommandos erwartet. Diese READY-Meldung signalisiert, daß Ihnen der Rechner zur Verfügung steht. Während des Programmablaufs erscheint demnach kein READY, und es können somit keine Eingaben gemacht werden.

Nun zu dem kleinen blinkenden Quadrat unterhalb des READY. Dies ist eine Orientierungshilfe auf dem Bildschirm. Angenommen, Sie möchten etwas auf dem Bildschirm schreiben, dann ist es notwendig, genau zu wissen, wo das einzugebende Zeichen erscheinen wird. Diese Markierung, die man in der Fachsprache Cursor ("körsa" gesprochen) nennt, hilft Ihnen also, #sich am Bildschirm zurechtzufinden.

<Cursor links/rechts>-Taste



Diese Taste befindet sich unten rechts auf der Tastatur. Sie ist beschriftet mit der Abkürzung für Cursor (<Crsr>) und den Pfeilen nach rechts und links. Mit dieser Taste können Sie nun den Cursor auf dem Bildschirm nach rechts oder links verschie-

ben. Betätigen Sie nun diese Taste 20 Mal, so wandert der Cursor 20 Stellen nach rechts und blinkt wieder geduldig unterhalb der ersten Eins von 38911.

Was aber, wenn der Cursor den rechten Rand dieser Zeile überschreitet? Probieren Sie dies selbst aus, und drücken Sie dazu nochmals 20 mal die <Cursor links/rechts>-Taste. Nach dem 20. Druck auf diese Taste erscheint der Cursor wieder auf der 1. Spalte der folgenden Zeile.

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE

 READY
                    ■
```

Es ist doch recht umständlich, 20 mal auf die Cursor-Taste zu hämmern, um in die Bildschirmmitte zu gelangen. Diese Prozedur können Sie etwas vereinfachen. Wenn Sie die Taste gedrückt halten, so wandert der Cursor selbständig mit 15 Zeichen je Sekunde nach rechts. Probieren Sie dies aus, und halten Sie dazu die <Cursor links/rechts>-Taste gedrückt. Beobachten Sie, wie der Cursor zum rechten Bildschirmrand "sprintet".

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE

 READY
 ■
```

Sicher haben Sie sich gefragt, wie der Cursor denn nun in die andere Richtung (nach links) zu bewegen ist. Dazu benutzen Sie die Taste <Shift>. Diese Taste schaltet die Funktion der Cursor-Taste auf <Cursor links> um.

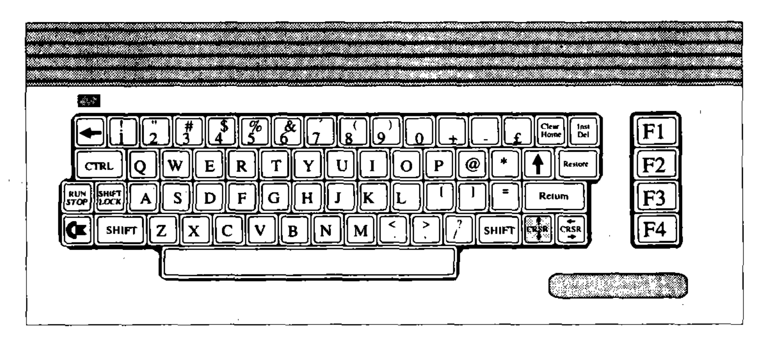
Halten Sie nun die <Shift>-Taste gedrückt und betätigen Sie die <Cursor links/rechts>-Taste. Der Cursor bewegt sich nun in die andere Richtung.

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE

READY
■
```

Selbstverständlich bewegt sich der Cursor auch selbständig nach links, ohne daß immer wieder einzeln auf die Taste gedrückt werden muß. Halten Sie dazu die beiden Tasten gedrückt.

<Cursor oben/unten>-Taste



Wie kann man nun den Cursor wieder in die Ausgangsstellung (unterhalb von READY) bringen? Dazu könnten Sie z.B. so lange die <Cursor links>-Taste gedrückt halten, bis der Cursor Zeile für Zeile nach oben wandert.

Doch dies ist nicht im Sinne des Erfinders. Die <Cursor oben/unten>-Taste ermöglicht Ihnen die ebenso schnelle Bewegung des Cursors nach oben und unten wie mit der <Cursor links/rechts>-Taste nach links und rechts.

Bewegen Sie nun den Cursor zum linken Bildschirmrand (<Shift> und <Cursor links/rechts> drücken).

Der Cursor blinkt wiederum geduldig, bis Sie ihn an eine andere Stelle bewegen. Peilen Sie nun die <Cursor oben/unten>-Taste an, und drücken Sie diese Taste dreimal. Sie haben bemerkt, daß der Cursor drei Zeilen nach unten gesprungen ist.

```
    **** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM  38911 BASIC BYTES FREE

READY


■
```

Versuchen Sie nun, den Cursor in die dritte Bildschirmzeile zu bewegen. Dazu müssen Sie den Cursor nach oben bewegen. Wenn Sie aber auf die <Cursor oben/unten>-Taste drücken, so bewegt sich der Cursor nach unten! Sicher haben Sie bereits geschaltet und Rückschlüsse aus der zuvor beschriebenen Taste gezogen. Hier wurde mit Hilfe der <Shift>-Taste der Cursor nach links bewegt. Genauso können Sie mit der <Shift>-Taste und der <Cursor oben/unten>-Taste den Cursor nach oben bewegen. Versuchen Sie es! Drücken Sie so lange <Shift> und <Cursor oben/unten>, bis der Cursor in der dritten Bildschirmzeile blinkt.

```
■   **** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM  38911 BASIC BYTES FREE

READY
```

Auch hier wandert der Cursor selbständig Zeile für Zeile nach oben oder nach unten, wenn Sie die <Cursor oben/unten>-Taste gedrückt halten.

Um ein Gefühl für die Cursor-Tasten zu erlangen, versuchen Sie einmal, mit dem Cursor ein unsichtbares Quadrat zu zeichnen. D.h., Sie bewegen den Cursor rechts→unten→links→oben oder anders herum links→unten→rechts→oben. Das Beherrschen der Cursor-Tasten ist erstrebenswert, da Sie später z.B. Korrekturen in einem Programm blitzschnell vornehmen können. Doch Übung macht den Meister, und der ist bekanntlich noch nicht vom Himmel gefallen.

Editieren mit den Cursor-Tasten

Hier taucht ein Fremdwort aus der Datenverarbeitung auf, dessen Bedeutung dem ein oder anderen Leser, besonders dem Anfänger, sicher nicht bekannt ist. Editieren bedeutet "Bearbeiten" von Texten, d.h., Texte erstellen und ändern. Man spricht bereits vom Editieren, wenn Sie ein paar Worte auf den Bildschirm schreiben.

Mit den Cursor-Tasten können Sie einen bereits auf dem Bildschirm enthaltenen Text ändern. Wenn Sie den Cursor auf ein Zeichen des Bildschirminhalts positionieren und eine beliebige Taste drücken, wird dieses Zeichen durch das zuletzt eingegebene Zeichen ersetzt.

Nun können wir z.B. die Einschaltmeldung verändern. Ändern wir die zweite Zeile der Einschaltmeldung um in "64 K RAM SYSTEM 38911 BYTES UNBENUTZT". Dazu bringen Sie zunächst den Cursor auf den ersten zu ändernden Buchstaben (das "A" von "BASIC").

```
    **** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM  38911 B■SIC BYTES FREE

READY
```

Dieses "A" soll nun durch ein "Y" ersetzt werden. Nichts leichter als das! Sie drücken einfach die Taste "Y", und das "A" ist verschwunden. Das "Y" hat nun dessen Platz eingenommen. Der Cursor ist selbständig ein Zeichen nach rechts gerückt. Das ist ja auch klar, sonst müßten Sie bei der Texteingabe nach jedem Zeichen den Cursor manuell weiterstellen.

```
    **** COMMODORE 64 BASIC V2 ****
64K RAM SYSTEM  38911 BY■IC BYTES FREE

READY
```

Ändern Sie nun auch die drei letzten Buchstaben dieses Wortes ("SIC") in "TES" um. Dazu geben Sie einfach unmittelbar hintereinander diese drei Buchstaben und anschließend ein Leerzeichen (die unübersehbare Taste ganz unten) ein. Nun haben wir das Wort "BASIC" in "BYTES" umgewandelt.

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BYTES ■YTES FREE

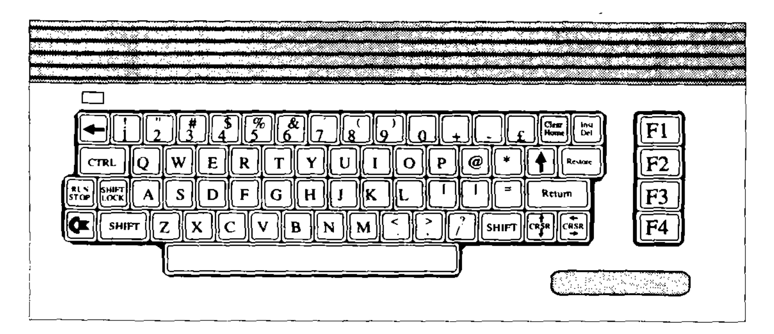
 READY
```

Der Abschluß dieser kleinen Übung ist für Sie sicher kein Rätsel mehr. Sie überschreiben den Rest der Zeile rücksichtslos mit dem Wort "UNBENUTZT".

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BYTES UNBENUTZT

 READY
```

<Clr/Home>-Taste



Wir haben uns bereits damit beschäftigt, den Cursor so schnell wie möglich zum oberen Bildschirmrand zu manövrieren. Dazu haben wir die <Cursor oben/unten>-Taste benutzt. Aber es geht noch schneller: Die <Clr/Home>-Taste hat zwei Funktionen: eine <Shift>- und eine Normalfunktion. Drücken Sie diese Taste ohne <Shift>, wird die Home-Funktion aktiv. Der Cursor wird dann

in die obere linke Ecke positioniert. Diese Ecke ist das "Zuhause" des Cursors. Überzeugen Sie sich davon, und drücken Sie nun die <Clr/Home>-Taste.

```
■   **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE

 READY
```

Damit Sie nicht aus der Übung kommen: Schreiben Sie ein paar Buchstaben in die Bildschirmmitte.

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE

 READY

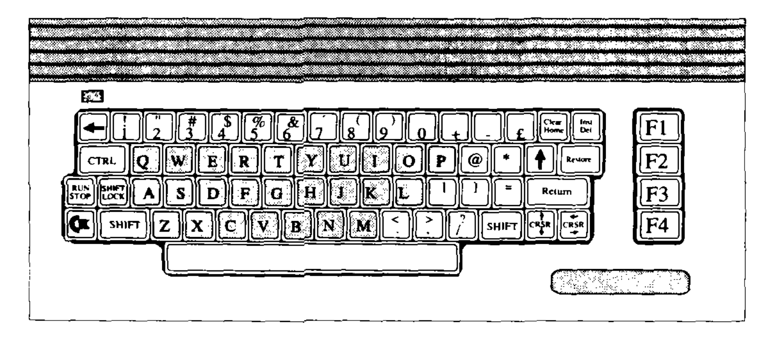
                ASDFGHJKL■
```

Wenn Sie nun denken, daß das, was sich nun auf dem Bildschirm befindet, eigentlich alles unbrauchbares Kauderwelsch ist, so wischen Sie einfach alles weg. Halt! Nicht mit dem Fensterleder, damit können Sie vielleicht den Staub vom Bildschirm wischen, nicht jedoch die eingetippten und somit im Rechner gespeicherten Daten des Bildschirms.

Schließlich ist es für einen Rechner, der ca. 1 Million Befehle pro Sekunde abarbeiten kann, ein Kinderspiel, "mal eben" die 1000 Zeichen auf dem Bildschirm auf Tastendruck zu löschen. Drücken Sie nun die Tasten <Shift> und <Clr/Home> gemeinsam, wird der Bildschirm blitzschnell gelöscht. Der Cursor blinkt dann ganz verlassen "zuhause", in der oberen linken Ecke des Bildschirms.

```
■
```

Die Tasten mit Buchstaben



Wie bereits erwähnt, gleicht die Anordnung dieser Tasten bis auf kleine Abweichungen der einer Schreibmaschine. Sicher sind Ihnen die jeweils zwei Grafiksymbole an der Vorderseite jeder dieser Tasten sofort aufgefallen. Doch, was es damit auf sich hat, stellen wir zunächst zurück.

Konzentrieren wir uns zunächst auf die Buchstaben. Wenn Sie irgendeine dieser Tasten drücken, so erscheint der entsprechende Großbuchstabe an der Cursor-Position auf dem Bildschirm. Bevor wir diese Tasten einsetzen, löschen Sie bitte den Bildschirm mit den Tasten <Shift> und <Clr/Home>.

Nun haben wir einen "sauberen" Bildschirm, auf dem wir editieren können. Bitte geben Sie nun keine Unmengen von Texten in der Hoffnung ein, daß Ihr Computer alles zu jeder Zeit abrufbereit in seinem Speicher aufnimmt. Ganz so einfach ist es nicht. Das, was Sie eingeben, wird lediglich im flüchtigen Bildschirmspeicher festgehalten. Wie man eingegebene Texte abspeichern kann, erfahren Sie in unserer BASIC-Einführung.

Schreiben Sie jetzt einmal alle Buchstaben auf den Bildschirm, vielleicht sogar in alphabetischer Reihenfolge. Sie werden feststellen, wie schwer doch zu Anfang der ein oder andere Buchstabe zu finden ist, wenn man nicht gerade Sekretärin ist. Auf dem Bildschirm befinden sich nun alle verfügbaren Großbuchstaben.

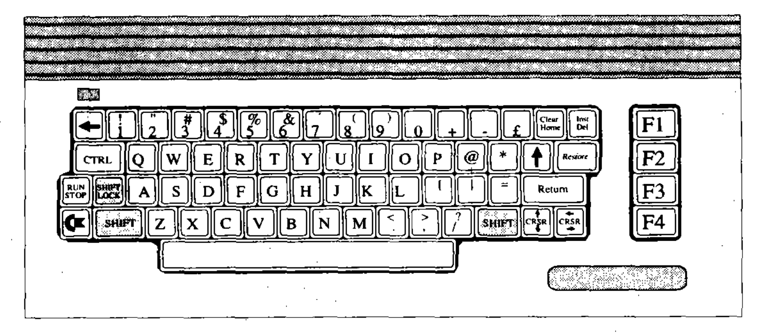
```
ABCDEFGHIJKLMNOPQRSTUVWXYZ■
```

Löschen Sie wieder den Bildschirm, und geben Sie nochmals alle Buchstaben ein. Doch nun nicht in alphabetischer Reihenfolge, sondern wie sie auf der Tastatur angeordnet sind.

Also zunächst die obere Reihe (Q bis P), dann die mittlere Reihe (A bis L) und schließlich die untere Reihe (Z bis M). Sicher fällt Ihnen diese Reihenfolge viel leichter.



Die <Shift>- und <Shift Lock>-Tasten



Die <Shift>-Taste haben Sie bereits kennengelernt. Sie benutzten diese Taste zum Umschalten der mit zwei Funktionen belegten Tasten. Zusätzlich ist der C64 mit einer zweiten <Shift>-Taste ausgerüstet (<Shift Lock>), die bei Betätigung einrastet.

Somit ist es überflüssig, die <Shift>-Taste gedrückt zu halten. Betätigen Sie diese Taste ein zweites Mal, wird sie wieder entriegelt.

Fast jede Taste ist mit mehr als einem Zeichen oder einer Funktion belegt. Mit der <Shift>-Taste erreichen Sie die zweite Funktion bzw. das zweite Zeichen einer mehrfach belegten Ta-

ste. Bei den Tasten, die zusätzlich mit zwei Grafiksymbolen beschriftet sind, wählen Sie mit der <Shift>-Taste das jeweils rechte Grafikzeichen aus.

Versuchen Sie nun, die vier Spielkartensymbole der Tasten "A","S","Z" und "X" auf den Bildschirm zu bekommen. Löschen Sie zuvor den Bildschirm mit den beiden Tasten <Shift> und <Clr/Home>.

Das Herz z.B. erscheint auf dem Bildschirm, wenn Sie die Taste <Shift> gedrückt halten und dann die Taste "S" betätigen.



Sicher ist es für Sie nun ein Kinderspiel, die anderen drei Symbole direkt neben dem einsamen Herzen auf den Bildschirm zu zaubern.

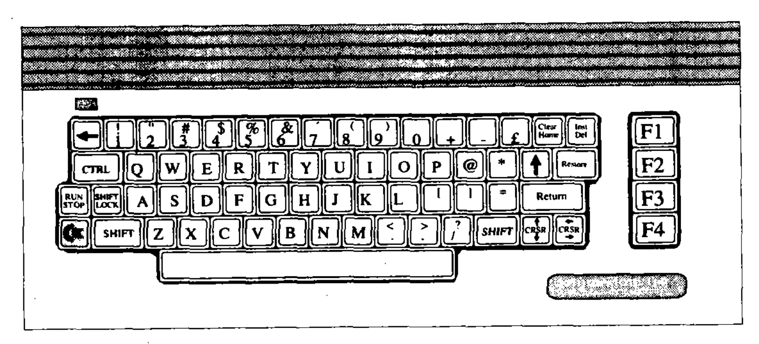
Das war noch relativ einfach. Aber nun eine etwas schwierigere Aufgabe: Stellen Sie nochmals alle Buchstaben, wie sie auf der Tastatur angeordnet sind, dar (vorher Bildschirm löschen). Dann positionieren Sie den Cursor auf den Anfang der nächsten Zeile und schreiben unter jeden Buchstaben das jeweils dazugehörige Grafikzeichen.

Ein Tip: Wenn Sie die <Shift Lock>-Taste einrasten, haben Sie bei der Eingabe der Grafikzeichen eine Hand frei, mit der Sie z.B. zur Kaffeetasse greifen können.

```
QWERTYUIOPASDFGHJKLZXCVBNM
```

Entriegeln Sie bitte die <Shift Lock>-Taste, wenn sie verwendet wurde.

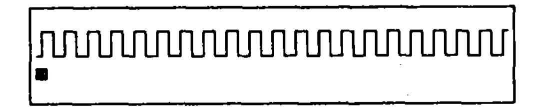
Die Commodore-Taste <C=>



Mehrfachbelegte Tasten sind für Sie nicht mehr unbekannt. Sie haben mit <Shift> z.B. das rechte Grafiksymbol eines Buchstabens zum Vorschein gebracht. Sie wissen sicher, worauf hier angespielt wird. Da die Buchstabentasten zwei Grafiksymbole darstellen können, muß natürlich auch eine Möglichkeit gegeben werden, das zweite Grafiksymbol anzusprechen. Dazu wurde die Commodore-Taste installiert. Halten Sie die Taste <C=> gedrückt, wenn Sie eines der links an der Vorderseite der Tasten dargestellten Grafiksymbole auswählen möchten. Nun aufgepaßt! Links neben den schon bekannten Spielkartensymbolen sind Grafikzeichen angebracht, die jeweils einen Winkel darstellen. Halten Sie nun die Taste <C=> gedrückt, und stellen Sie die zwei Winkel der Tasten "A" und "S" auf dem Bildschirm dar. Bitte löschen Sie auch hier wieder vorher den Bildschirm.



Nun eine etwas schwierigere Aufgabe: Versuchen Sie, das in Bild 2 dargestellte Muster nachzuzeichnen. Nein, nicht mit Bleistift und Papier, sondern mit den Tasten Ihres Rechners auf dem Bildschirm. Ein Tip: Dieses Muster besteht aus zwei Zeilen!



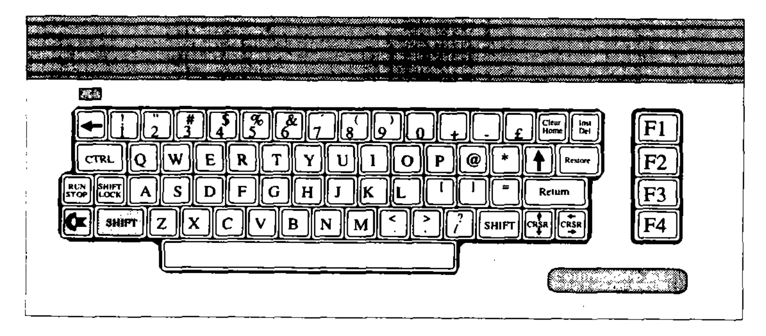
Schreiben Sie nun die drei mit der Taste "A" darstellbaren Zeichen nebeneinander auf dem Bildschirm. Zuerst den Buchstaben, dann das rechte und schließlich das linke Grafikzeichen. Löschen Sie bitte vorher den Bildschirm.



Mit einiger Übung können Sie allein mit den Grafikzeichen hübsche Bilder auf den Bildschirm zaubern. Vielleicht starten Sie in Ihrer Familie einen Wettbewerb nach dem Motto "Wer tastet das schönste Bild auf den Bildschirm?".

Diese einfarbigen Bilder können Sie auch noch in 16 Farben darstellen, doch dazu erfahren Sie an späterer Stelle mehr.

Der Textmodus



Die beiden Tasten <Shift> und <C=> haben zusammen noch eine weitere Aufgabe: Sie ermöglichen die Groß-/und Kleinschreibung. In diesem Modus gleicht die Tastatur noch mehr einer Schreibmaschine. Bei normaler Betätigung werden Kleinbuchstaben und mit Hilfe der Umschalt-Taste (bei uns die <Shift>-Taste) Großbuchstaben angezeigt. Die Grafiksymbole, die normalerweise mit <Shift> angesprochen werden, stehen nun nicht mehr zur Verfügung. Sie schalten den Textmodus ein, indem Sie die beiden Tasten <Shift> und <C=> gleichzeitig drücken. Wollen Sie wieder im Normalmodus arbeiten, so drücken Sie nochmals diese beiden Tasten. Schalten Sie nun den Textmodus ein und

beobachten Sie, was aus den drei Zeichen der letzten Übung geworden ist. Sie werden sofort dem Textmodus angepaßt. Aus dem großem "A" wurde ein kleines "a", aus dem PIK wurde ein großes "A" und das zweite Sonderzeichen ist erhalten geblieben.



Es gibt also zwei verschiedene Möglichkeiten, die Tastatur zu nutzen:

1. Den Groß/Grafikmodus.
2. Den Groß/Kleinmodus.

Wollen Sie also Kleinbuchstaben darstellen, so müssen Sie auf die Grafikzeichen, die mit <Shift> angesprochen werden, verzichten und umgekehrt.

Schreiben Sie nun einmal im Textmodus den Namen "Karl Schmidt" auf den Bildschirm. Die Großbuchstaben werden mit der <Shift>-Taste eingegeben.

```
Karl Schmidt
```

Was meinen Sie, tut sich auf dem Bildschirm, wenn Sie nun den Textmodus wieder ausschalten? Überzeugen Sie sich selbst. Schalten Sie den Textmodus wieder aus, indem Sie wiederum <Shift> und <C=> gleichzeitig drücken.

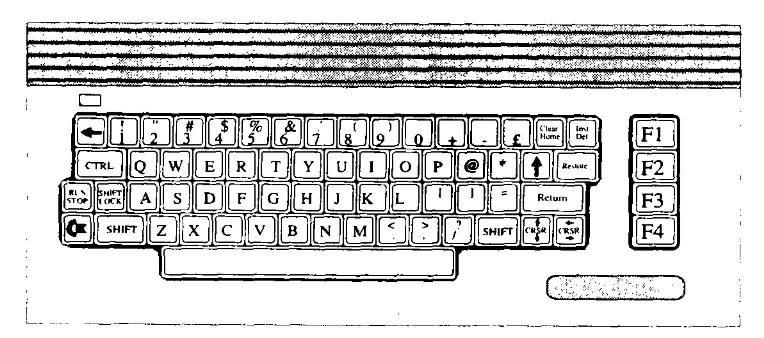
```
ARL ♠CHMIDT
```

Aus den Großbuchstaben wurden wieder Grafikzeichen und aus den Kleinbuchstaben wurden Großbuchstaben.

In der Praxis hat sich erwiesen, daß man im Textmodus besser arbeiten kann. Der Text erscheint in Kleinbuchstaben nicht so gedrängt wie in Großbuchstaben. Programmlisten z.B. sind we-

sentlich übersichtlicher, wenn sie im Textmodus auf dem Bildschirm angezeigt werden. Ein ständig eingeschalteter Textmodus ist selbstverständlich nur geeignet, wenn ohne Grafikzeichen gearbeitet wird.

Weitere Grafiktasten

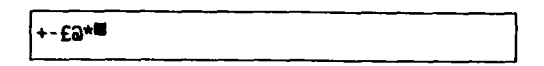


Außer den vielen Grafikzeichen auf den Buchstabentasten finden Sie weitere Grafikzeichen auf diesen fünf Tasten. Sie werden entsprechend mit den Tasten <Shift> und <C=> angesprochen. Die Tasten +, -, £, @ und * haben jedoch eine Besonderheit: Sie sind sowohl im Textmodus als auch im Normalmodus identisch.

Überzeugen Sie sich davon, und geben Sie die Zeichenfolge "+-£@*" ein, nachdem Sie den Bildschirm gelöscht haben.



Schalten Sie nun den Textmodus ein, so werden Sie feststellen, daß die Zeichen unverändert geblieben sind.



Verlassen Sie wieder den Textmodus (wiederum mit <Shift> und <C=>). Im Textmodus werden bekanntlich aus den <Shift>-Grafikzeichen Großbuchstaben. Da auch die hier behandelten fünf

Tasten <Shift>-Grafikzeichen enthalten, geben wir diese Zeichen ein, um festzustellen, ob der Textmodus hier etwas verändert. Löschen Sie also nochmals den Bildschirm und betätigen Sie diese fünf Tasten, diesmal zusammen mit <Shift>.

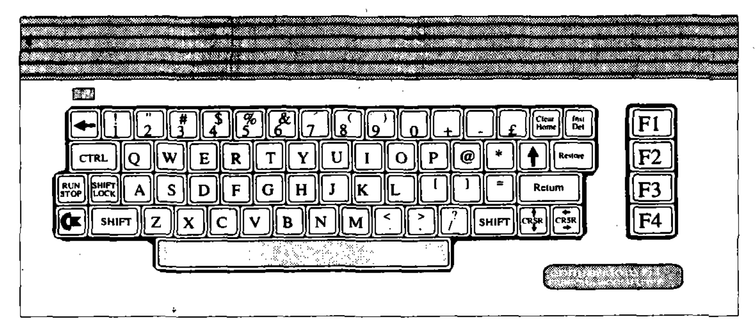


Wenn Sie nun den Textmodus einschalten, werden Sie feststellen, daß zwei Tasten im Textmodus andere Grafikzeichen zum Vorschein bringen. Dies sind zwei Zeichen, die Sie auf der Tastatur zwar nicht finden, die aber im Textmodus plötzlich erscheinen.

+ | ⁄⁄ — ▒

Schalten Sie den Textmodus bitte aus, bevor Sie fortfahren.

Die Leertaste



Auch diese Taste finden Sie an jeder Schreibmaschine wieder. Auch bei Ihrem Computer brauchen Sie nicht darauf zu verzichten, Leerzeichen zwischen die Wörter zu setzen. Befindet sich an der Stelle, an der ein Leerzeichen erscheinen soll, bereits ein anderes Zeichen, so wird dieses natürlich gelöscht. Löschen Sie den Bildschirm, und geben Sie den folgenden Satz ein:

```
UEBUNG MACHT DEN MEISTER
```

Hier müssen Sie zwischen den einzelnen Wörtern die Leertaste drücken. Sie können auch den Cursor um eine Stelle nach rechts rücken, doch dies ist sehr umständlich und somit nicht angebracht.

```
UEBUNG MACHT DEN MEISTER
```

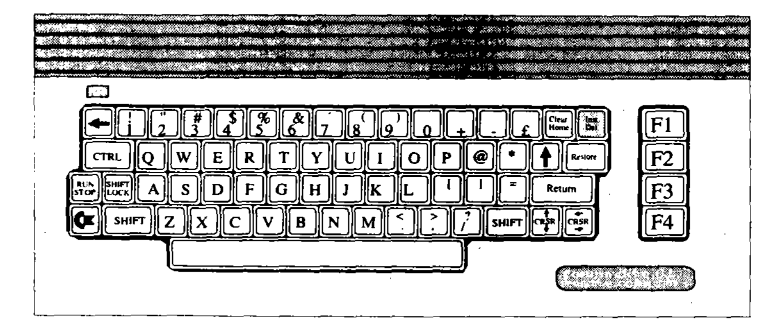
Mit der Leertaste können auch Zeichen von links nach rechts gelöscht werden. Versuchen Sie es selbst: Drücken Sie die Taste <Clr/Home>, um zum linken Rand der obersten Bildschirmzeile zu gelangen.

```
■EBUNG MACHT DEN MEISTER
```

Halten Sie nun die Leertaste gedrückt, so werden alle eingegebenen Zeichen durch Leerzeichen ersetzt, also gelöscht.

```
                      ■STER
```

Die <Inst/Del>-Taste



Die Abkürzungen dieser Taste sagen bereits viel über die Funktion aus. <Del> bedeutet Delete (Löschen) und <Inst> bedeutet Insert (Einsetzen). Sie verfügen somit über zwei komfortable Hilfen zum Editieren von Texten.

<Del> - Löschen

Ohne <Shift> ist die Funktion DEL aktiv. Im letzten Beispiel haben wir bereits mit der Leertaste Zeichen auf dem Bildschirm gelöscht. Doch dies war nur in einer Richtung - nach rechts - möglich. Mit DEL können nun Zeichen links vom Cursor gelöscht werden. Dies ist besonders hilfreich beim Korrigieren von Tippfehlern. Ein Druck auf die Taste <Inst/Del>, und das falsch eingetippte Zeichen ist wieder gelöscht. Löschen Sie nun den Bildschirm, und tippen Sie den folgenden Satz ab:

```
TIPPFEHLER SIND UNVERZEIHLICJ
```

Bei der Eingabe des letzten Buchstabens ist ein Fehler passiert:

```
TIPPFEHLER SIND UNVERZEIHLICJ■
```

Tipp-Ex ist hier nicht notwendig. Auf Tastendruck wird der zuletzt eingegebene Buchstabe gelöscht. Drücken Sie dazu die Taste <Inst/Del>.

```
TIPPFEHLER SIND UNVERZEIHLIC■
```

Nun können Sie hier den richtigen Buchstaben einsetzen, und der Tippfehler ist vergessen.

```
TIPPFEHLER SIND UNVERZEIHLICH■
```

Doch dieses Beispiel zeigt nicht den überragenden Vorteil dieser Taste. Spielen wir ein weiteres Beispiel durch. Geben Sie, nachdem Sie den Bildschirm gelöscht haben, den folgenden Satz ein:

```
TIPPFEHLER SIND UNVERRZEIHLICH
```

Auch hier hat sich ein Tippfehler eingeschlichen: Zwei "R" sind hier nicht angebracht.

```
TIPPFEHLER SIND UNVERRZEIHLICH■
```

Es ist nicht notwendig, das letzte Wort ab dem Tippfehler nochmal zu schreiben. Wird nämlich ein Zeichen links vom Cursor mit der Taste <Inst/Del> gelöscht, so werden automatisch alle Zeichen in dieser Zeile, die sich rechts vom Cursor befinden, nachgerückt. Das ist nicht einfach zu erklären, sehen Sie darum selbst:

Bewegen Sie den Cursor auf das zu löschende "R" des fehlerhaften Satzes. Dazu drücken Sie die Tasten <Shift> und <Cursor links/rechts> gleichzeitig.

```
TIPPFEHLER SIND UNVER■ZEIHLICH
```

Wenn Sie nun die Taste <Inst/Del> drücken, wird das "R" links neben dem Cursor gelöscht und der Rest einschließlich des Zeichens unter dem Cursor nach links aufgerückt.

```
TIPPFEHLER SIND UNVE■ZEIHLICH
```

Wenn Sie nun den Cursor zum Ende dieses Satzes bringen, können Sie mit dem Schreiben fortfahren. Halten Sie dazu so lange die <Cursor links/rechts>-Taste gedrückt, bis der richtige Punkt erreicht ist.

```
TIPPFEHLER SIND UNVERZEIHLICH
```

<Inst> - Einfügen

Die <Inst/Del>-Taste hat wie gesagt zwei Funktionen: Einmal - wie bereits beschrieben - die DEL-Funktion, mit der Sie Zeichen links vom Cursor löschen können. Die zweite Funktion wird wie üblich mit der <Shift>-Taste aktiviert. Die Einfügefunktion Ihres C64 ist sehr beliebt, doch überzeugen Sie sich

selbst an dem folgenden Beispiel. Geben Sie zunächst den folgenden Satz ein, nachdem Sie den Bildschirm gelöscht haben:

```
ICH MACHE NIMALS FEHLER
```

Hierin ist ein Fehler enthalten, der mit der <Inst/Del>-Taste wieder korrigiert werden kann.

```
ICH MACHE NIMALS FEHLER■
```

Jedem, der in der Schule ein wenig aufgepaßt hat, wird dieser Fehler sicher sofort ins Auge gefallen sein. Es wäre nun sehr umständlich, wenn man alle Zeichen bis zum Fehler löschen und neu eingeben müßte. Da gerade diese Einfügefunktion sehr oft benötigt wird, ist sie mit den Tasten <Shift> und <Inst/Del> sofort abrufbereit. Bewegen Sie dazu den Cursor hinter das Zeichen, nach dem weitere Zeichen eingefügt werden sollen, in unserem Fall hinter das "I", also auf das "M".

```
ICH MACHE NI■ALS FEHLER
```

Beobachten Sie nun, was passiert, wenn Sie die <Shift>-Taste gedrückt halten und die <Inst/Del>-Taste einmal betätigen. Es wird Platz für das einzufügende Zeichen geschafft wobei der Cursor auf diesem Platz verbleibt.

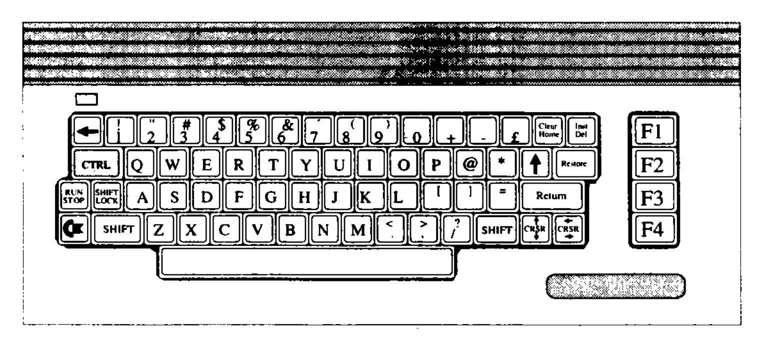
```
ICH MACHE NI■MALS FEHLER
```

Nun geben Sie das gewünschte Zeichen ein, in unserem Beispiel also das fehlende "E". Wenn Sie den Cursor nun nach rechts zum Ausgangspunkt bewegen, können Sie die Texteingabe fortsetzen.

Selbstverständlich können Sie auch mehrere Zeichen einfügen. Dazu drücken Sie an der entsprechenden Positition mehrmals die <Inst/Del>- mit der <Shift>-Taste. Wenn Sie beide Tasten gedrückt halten, tritt eine automatische Wiederholung in Kraft.

Die Taste <Inst/Del> wird dann selbständig so lange betätigt, bis Sie beide Tasten wieder loslassen. Gerade die Bedienung der Taste <Inst/Del> bereitet am Anfang Schwierigkeiten. Doch Sie werden selbst feststellen, daß Sie mit der Zeit sehr viel Routine beim Umgang mit dieser Taste entwickeln werden.

Die <Ctrl>-Taste bringt Farbe ins Spiel



Sicher ist Ihnen bekannt, daß Ihr C64 16 Farben darstellen kann. Sicher warten Sie auch schon lange darauf, diese selbst hervorzuzaubern. Nichts leichter als das. Jede der Zifferntaste 1 bis 8 schaltet zusammen mit der <Ctrl>-(Control-)Taste auf eine andere Zeichenfarbe um. Halten Sie z.B. <Ctrl> gedrückt und tippen auf die Taste <1>, so schreiben Sie ab sofort in schwarzer Schrift.

Die acht Tasten und die zugehörigen Farben:

Taste	Beschriftung	Farbe
<1> BLK	black	schwarz
<2> WHT	white	weiß
<3> RED	red	rot
<4> CYN	cyan	türkis
<5> PUR	purple	violett
<6> GRN	green	grün
<7> BLU	blue	blau
<8> YEL	yellow	gelb

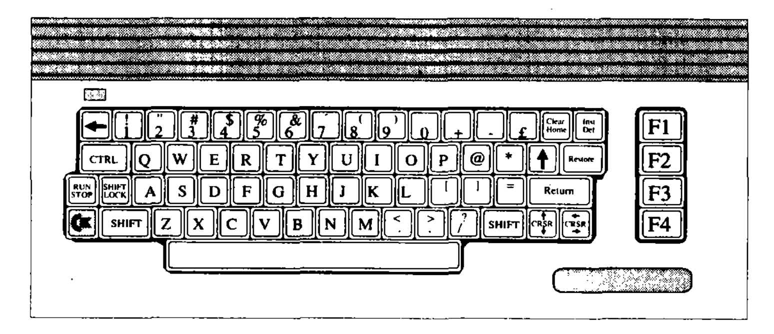
Wenn Sie nun alle Farben austesten, so werden Sie feststellen, daß die ein oder andere Farbe auf dem blauen Hintergrund schlecht lesbar ist. Man sagt dann, der Kontrast stimmt nicht.

Doch dies ist nicht weiter tragisch, da Sie die Farbe des Hintergrunds mit einem BASIC-Befehl, den Sie an späterer Stelle noch kennenlernen werden, ändern können.

Probieren Sie nun einmal folgendes aus: Drücken Sie die <Ctrl>-Taste zusammen mit der Taste <7>. Was passiert und wie ist dies zu erklären? Ganz einfach: Die Farbe der Taste <7> ist die Farbe blau und somit mit dem Hintergrund identisch.

Der Cursor ist wie alle anderen Zeichen zwar vorhanden, aber unsichtbar. Man könnte nun sogar Befehle eingeben, die man aber auf dem Bildschirm nicht sieht. Innerhalb von Programmen nutzt man dies oft aus, um Meldungen des Rechners für den Benutzer unsichtbar zu machen.

Weitere Farbenpracht mit der <C= >-Taste



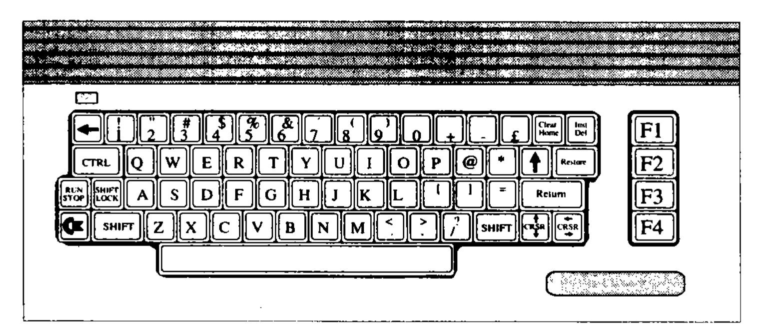
Viele vergleichbare Computer sind zumeist mit acht Farben ausgerüstet. Doch Ihr C64 gibt sich damit nicht zufrieden. Mit der <Commodore>-Taste und den Zifferntasten 1 bis 8 können weitere acht brillante Farben ausgewählt werden. Diese Farben sind aber auf den Tasten nicht ausgewiesen. Drücken Sie nun einmal die <C=>-Taste und die Taste <1> gleichzeitig. Der Cursor wechselt zur Farbe "Orange", und alle Zeichen, die Sie nun ein-

geben, erscheinen ebenfalls in der Farbe Orange. Der folgenden Tabelle können Sie alle 16 Farben und die dazu zu bedienenden Tasten entnehmen:

Taste	Beschriftung	<Ctrl> <C=>
<1> BLK	schwarz	orange
<2> WHT	weiß	braun
<3> RED	rot	hellrot
<4> CYN	türkis	grau 1
<5> PUR	violett	grau 2
<6> GRN	grün	hellgrün
<7> BLU	blau	hellblau
<8> YEL	gelb	grau 3

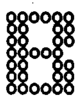
Versuchen Sie nun einmal, die ersten 16 Zeilen des Bildschirms mit jeweils einer Farbe und dem Zeichen "A" zu füllen. Achten Sie auf den rechtzeitigen Farbwechsel. Die Zeile mit der Farbe "Blau" ist natürlich nicht zu erkennen, da Sie mit der Hintergrundfarbe identisch ist.

Reverse Zeichendarstellung



Die deutsche Übersetzung für "REVERSE" ("rivörs" gesprochen) ist sinngemäß "UMGEKEHRT". Doch was sind "umgekehrte" Zeichen? Sicher haben Sie schon einmal Negative von Schwarzweiß-Fotos betrachtet und festgestellt, daß die Farben vertauscht sind. Ein dunkler Hintergrund des Fotos ist auf dem Negativ hell.

Wenn Sie die Zeichen auf dem Bildschirm genau betrachten, so merken Sie, daß sie aus einzelnen Punkten zusammengesetzt sind. Alle Zeichen werden in einer Matrix von 8 mal 8 Bildschirmpunkten dargestellt. Der Buchstabe "B" z.B. sieht dann etwa so aus:



Wenn dieses Zeichen nun "reverse" dargestellt wird, werden alle gesetzten Punkte der 8x8-Matrix gelöscht und umgekehrt. Das "B" sieht demnach dann so aus:

Wenn Sie bei der Farbauswahl zufällig die Taste <9> zusammen mit <Ctrl> betätigt haben, sind Sie schon unfreiwillig mit den reversen Zeichen konfrontiert worden. Die reverse Zeichendarstellung wird mit den Tasten <Ctrl> und <9> eingeschaltet.

Löschen Sie nun den Bildschirm, und schreiben Sie in die erste Zeile den Text

```
REVERSE IST NUN AUSGESCHALTET
```

Nachdem Sie den Cursor zum Anfang der zweiten Zeile befördert haben, halten Sie bitte die Taste <Ctrl> gedrückt und betätigen kurz die Taste <9>. Schreiben Sie danach den folgenden Text auf den Bildschirm:

```
REVERSE IST NUN EINGESCHALTET
```

Jetzt sehen Sie bestens, was reverse Zeichendarstellung bedeutet. Natürlich muß dies auch wieder auszuschalten sein. Die Taste rechts neben der <9>, also die Taste <0> schaltet diesen Reversemodus zusammen mit <Ctrl> wieder aus. Halten Sie dazu die <Ctrl>-Taste gedrückt, und tippen Sie kurz auf die Taste <0>. Sie können in der gewohnten Art und Weise weiterschreiben.

```
REVERSE IST NUN AUSGESCHALTET
REVERSE IST NUN EINGESCHALTET
```

Wozu benötigt man diese ungewohnte Zeichendarstellung? Eine berechtigte Frage, die jedoch leicht zu beantworten ist. Wenn Sie schon einmal Programme aus dem Handel gestartet und bedient haben, werden Sie sicher hier und da reverse Zeichen erkannt haben. Z.B. wird die letzte Bildschirmzeile gerne dazu benutzt, den Benutzer des Programms auf Fehler hinzuweisen. Diese Fehlermeldungen werden dazu "reverse" ausgegeben, damit sie sofort ins Auge fallen. Eine weitere Möglichkeit ist, die ersten ein oder zwei Buchstaben von aufgelisteten Teilprogrammen "reverse" darzustellen. Der Benutzer wählt dann das entsprechende Teilprogramm aus, indem er die zugehörigen reversen Zeichen eingibt. Ein Beispiel:

```
ADRESSENVERWALTUNG

PROGRAMMFUNKTIONEN

EINGEBEN
AENDERN
LOESCHEN
SUCHEN

AUSWAHL?
```

Hier sind die jeweils ersten Zeichen der auszuwählenden Teilprogramme "reverse" dargestellt. Sie geben nun einen der Buchstaben "E", "A", "L" oder "S" ein, und das gewünschte Teilprogramm wird aktiv. Übrigens: So etwas nennt man in der Fachsprache "MENÜ", in diesem Fall ein "MENÜ" in vier Gängen.

Tips zum Computerarbeitsplatz

Wahrscheinlich haben Sie Ihren Commodore 64 jetzt gerade auf dem Wohnzimmertisch stehen und die verschiedenen Anschluß-

kabel laden zum Darüberstolpern ein. Das ist natürlich nicht gerade der ideale Computerarbeitsplatz.

Wenn es sich irgendwie einrichten läßt, sollten Sie schauen, daß Sie einen dauerhaften Platz für Ihre Anlage finden. Ein kleiner Tisch von etwa 1,20 Meter Länge und 0,60 Meter Breite reicht dazu schon aus. Falls Sie nicht auf den Familienfernseher angewiesen sind oder über einen eigenen Monitor verfügen, sollten Sie den Bildschirm direkt hinter den Commodore 64 stellen. Ideal ist ein zusätzlicher Monitorständer, der sich drehen und kippen läßt. Solche Ständer sind mit etwa 30 Mark nicht allzu teuer und in jedem Fall eine lohnende Anschaffung.

Rechts und links neben dem Commodore 64 sollten Sie ausreichend Platz lassen. Auf der rechten Seite brauchen Sie evtl. Platz für einen Joystick oder eine Maus. Auf der linken Seite sollte sich zumindest ein aufgeschlagenes Buch, wie zum Beispiel dieses, unterbringen lassen.

Falls Sie über eine Datasette oder eine Floppy verfügen, so stellen Sie diese am besten etwas abseits, auf keinen Fall aber unmittelbar neben den Fernseher oder den Monitor. Insbesondere ältere Fernsehgeräte strahlen unter Umständen Störfelder aus, die zu einem Datenverlust auf der Kassette oder Diskette führen können. Überhaupt sollten Sie alles, was evtl. magnetische Felder ausstrahlt, weit aus dem Umkreis Ihrer Datasette oder Floppy bannen.

Daß Ihr Commodore 64 auf Flüssigkeiten jeder Art sehr allergisch reagiert, braucht wohl nicht extra erwähnt zu werden. Kaffeetassen, Colas und ähnliches gehören daher auf keinen Fall auf den Computertisch. Schneller als man denkt hat man ein Glas umgestoßen! Eine unter Umständen sehr teuere Reparatur ist dann die unvermeidliche Folge.

Ein weiterer wichtiger Punkt ist die Sonnenbestrahlung bzw. die Beleuchtung. Wie alle elektronischen Geräte reagiert der Commodore 64 auch auf zu starke Hitzeeinwirkung sehr empfindlich. Man sollte das Gerät daher nicht so aufstellen, daß es stundenlang der Sonne ausgesetzt ist. Auf der anderen Seite sollten Sie

auf eine ausreichende Beleuchtung des Arbeitsplatzes achten. Eine kleine Steh- oder Tischlampe eignet sich dazu am besten. Ganz wichtig: Die Lampe nicht so aufstellen, daß sie direkt auf den Bildschirm strahlt! Als guter Platz erweist sich eine Position rechts oder links neben dem Bildschirm.

Ein kleiner Tip am Rande: Auch wenn Sie für Ihre Commodore-64-Anlage ein dauerhaftes Plätzchen gefunden haben, werfen Sie auf keinen Fall die ganzen Verpackungen weg! Wenn Sie die Geräte später einmal wieder transportieren müssen (etwa zur Reparatur) werden Sie froh sein, wenn Sie noch die Originalverpackung haben.

1.2 Fernseher oder Monitor?

Einen Fernseher wird wohl jeder zu Hause herumstehen haben. Daher dürfte ein Fernseher das erste Sichtgerät sein, an das man seinen Commodore 64 anschließt. Spätestens wenn man mit schmerzenden und tränenden Augen vor dem Gerät sitzt, wird man sich aber fragen, ob es nicht etwas besseres gibt. Und das gibt es in der Tat: einen Monitor!

Im Gegensatz zu einem Fernseher ist ein Monitor ganz auf den Betrieb an einem Computer ausgerichtet. Das zeigt sich unter anderem darin, daß er über kein Antennenemfangsteil verfügt. Allerdings gibt es seit einiger Zeit für etwa 200 bis 300 Mark Zusatzgeräte, mit denen sich ein Monitor zu einem Fernseher erweitern läßt. Bedenkt man aber, daß es gute Fernsehportables heutzutage schon für um die 500 Mark gibt, so dürfte die Anschaffung eines derartigen Zusatzgerätes kaum lohnend sein.

Welche Arten von Monitoren gibt es nun?

Da sind zunächst die sogenannten Monochrom-Monitore. Diese Monitore erlauben nur eine einfarbige Darstellung, entweder in grün, weiß oder bernstein. Die tatsächlich eingestellten Farben werden aber in entsprechenden Farbabstufungen dargestellt. Bei einem Grün-Monitor erscheint beispielsweise die Farbe blau dann als mittelgrün und die Farbe weiß als hellgrün. Solange ein

Programm nicht zu viele Farben verwendet, hat man daher auch auf einem Monochrom-Monitor eine ausreichend kontrastreiche Darstellung.

Gute Monochrom-Monitore gibt es schon ab etwa 250 Mark. Beim Kauf sollten Sie darauf achten, daß Sie ein Gerät mit einem sogenannten Composite-Eingang bekommen. Diese Geräte können Sie bei einem späteren Systemwechsel problemlos auch an einen Amiga oder einen PC anschließen. Im Gegensatz zu den Monochrom-Monitoren erlauben Farbmonitore eine farbige Darstellung, wie am Fernseher, nur eben mit einer höheren Auflösung und einem flimmerfreieren Bild.

Probleme gibt es allerdings beim Anschluß an den Commodore 64. Professionelle Geräte aus dem PC-Bereich können in der Regel am Commodore 64 nicht ohne größere Umstände angeschlossen werden. Diese dürften allerdings auch schon wegen ihres hohen Preises (zum Teil über 2000 Mark) von vorneherein ausscheiden.

Für etwa 600 Mark wird von Commodore selbst ein Farbmonitor für den Commodore 64 angeboten, der über ein recht gutes Bild und zusätzlich über einen Stereoton-Ausgang verfügt. Nun stellt sich natürlich die Frage, soll man sich einen Monitor zulegen und wenn ja, welchen?

Eine generelle Antwort auf diese Frage läßt sich nicht geben. Letztendlich kommt es darauf an, mit welchen Anwendungen Sie auf dem Commodore 64 arbeiten möchten. Wenn Sie sich beispielsweise hauptsächlich für grafische Anwendungen oder für Spiele interessieren, sollten Sie sich einen Farbmonitor zulegen. Wenn Sie dagegen mehr "ernsthafte" Anwendungen, wie etwa Textverarbeitung oder Programmieren, betreiben möchten, empfiehlt sich die Anschaffung eines Monochrom-Monitors. Am "augenfreundlichsten" erweist sich dabei ein sogenannter Grün-Monitor.

Allerdings sollte man - auch mit dem besten Monitor ausgerüstet - nicht zu lange vor dem Computer sitzen. Ihre Augen werden es Ihnen danken!

1.3 Datasette und Floppy

Egal, ob Sie nur mit fertiger Software arbeiten oder auch selbst programmieren wollen, Sie benötigen ein (dauerhaftes) Speichermedium. Eine Möglichkeit ist die Anschaffung einer sogenannten Datasette. Eine Datasette ist im Grunde genommen nichts anderes als ein umgebauter Kassettenrekorder. Anstelle der Musik werden eben Computerdaten aufgezeichnet bzw. wiedergegeben.

Eine Datasette ist sehr günstig (für etwa 50-70 Mark) zu haben. Um es aber gleich vorweg zu sagen: Wenn Sie auch nur halbwegs vernünftig mit Ihrem Commodore 64 arbeiten möchten, werden Sie früher oder später nicht um den Kauf einer sogenannten Floppy herumkommen. Diese ist mit knapp 400 Mark allerdings nicht gerade billig.

Eine Floppy, auch Diskettenlaufwerk genannt, ist im weiteren Sinne mit einem Plattenspieler vergleichbar. Anstelle von Schallplatten benötigt man sogenannte Disketten. Das sind kleine, magnetisch beschichtete Scheiben, die sich aber im Gegensatz zur Schallplatte beliebig oft beschreiben lassen. Die Diskette dreht sich in der Floppy mit sehr hoher Geschwindigkeit und erlaubt daher einen schnellen Zugriff auf die gespeicherten Daten.

Hier liegt auch der wesentliche Unterschied zur Datasette. Mit einer Datasette können die Daten nur sequentiell, d.h. der Reihe nach, gelesen werden. Wenn Sie sich zum Beispiel gerade am Anfang einer Kassette befinden und die zu ladenden Daten ganz am Ende stehen, kann es bis zu einer halben Stunde dauern, bis die Daten geladen werden! Bei der Floppy dagegen ist es völlig egal, an welcher Stelle auf der Diskette die Daten stehen. Jeder Punkt der Diskette läßt sich in Sekundenschnelle ansteuern. Ein weiteres Problem bei der Datasette ist die langsame Übertragungsgeschwindigkeit. Diese beträgt nur etwa 300 Baud, das sind etwa 60 Zeichen je Sekunde, d.h., in einer Sekunde werden nur etwa 60 Zeichen von der Datasette in den Speicher des Commodore 64 geladen. Bei der Floppy ist die Übertragungsgeschwindigkeit über zehnmal so hoch! Auch das längste Programm wird daher in wenigen Minuten in den Rechner geholt.

Natürlich stellt eine Anschaffung in Höhe von rund 400 Mark keine Kleinigkeit dar. Manch einer wird daher notgedrungen zunächst mit einer Datasette arbeiten müssen. Für all diejenigen gibt es, sozusagen als Trostpflaster, das Kapitel 10. In Kapitel 10 erfahren Sie, wie man aus seiner Datasette durch geschickte Kniffe und Tricks wirklich das Letzte herausholen kann.

1.4 Der richtige Drucker

Wenn Sie die Ergebnisse Ihrer Arbeit am Commodore 64 "Schwarz auf Weiß" haben wollen, benötigen Sie einen Drucker. Was die Preise anbelangt, reicht das Spektrum von rund 300 Mark bis weit über 2000 Mark. Des hohen Preises sollte man den Druckerkauf sehr sorgfältig planen. Entscheidend ist, für welche Art von Ausdrucken Sie den Drucker benötigen.

Wenn Sie den Drucker nur für den "Hausgebrauch" einsetzen wollen, also zum Beispiel nur gelegentlich Programm-Listings ausdrucken möchten, dann sind die Druckqualität und die Druckgeschwindigkeit nicht so wichtig. In diesem Fall reicht ein einfacher 9-Nadel-Matrixdrucker für 300 bis 400 Mark.

Wenn Sie dagegen mit dem Commodore 64 beispielsweise Briefe schreiben möchten, dann sollen diese natürlich auch in ansprechender Qualität und nicht zu langsam zu Papier gebracht werden. In diesem Fall empfiehlt sich der Kauf eines 24-Nadel-Matrixdruckers, der allerdings mit knapp 1000 Mark zu Buche schlägt.

Ein anderes Problem ist der Anschluß des Druckers an den Commodore 64. Die von Commodore selbst angebotenen Drucker für den Commodore 64 verfügen über eine spezielle Schnittstelle, mit der man den Drucker direkt an den Rechner anschließen kann. Dafür verträgt sich diese Schnittstelle aber mit keinen anderen Computern, d.h., Sie können diesen Drucker nur am Commodore 64 betreiben. Das mag zwar für den Augenblick ausreichen, wenn Sie aber später auf einen anderen Rechner umsteigen, können Sie mit dem Drucker nichts mehr anfangen.

Professionelle Drucker, wie die angesprochenen 24-Nadel-Matrixdrucker, verfügen in der Regel über eine sogenannte Centronics-Schnittstelle, mit der sie problemlos an alle Personalcomputer (PC), aber eben nicht an den Commodore 64 angeschlossen werden können. In diesem Fall hilft ein sogenanntes Interface, das zwischen den Drucker und den Rechner geschaltet wird. Solch ein Interface ist separat erhältlich und kostet bis zu etwa 100 Mark.

Dank der weiten Verbreitung des Commodore 64 sind einige Druckerhersteller in der Zwischenzeit dazu übergegangen, ihre Drucker werksmäßig mit einer zusätzlichen Commodore-64-Schnittstelle auszustatten. Der Aufpreis beträgt dann meist nicht mehr als etwa 50 Mark. So gibt es zum Beispiel den EPSON LX 800 komplett mit Centronics- und Commodore-64-Interface für knapp 550 Mark.

1.5 Joysticks und Mäuse

Zwei sehr nützliche und im Gegensatz zu Monitor, Floppy und Drucker relativ preisgünstige Peripheriegeräte sind der sogenannte Joystick und die Maus.

Der Joystick ist natürlich ein unentbehrliches Hilfsmittel für jeden Spiele-Programmierer. Hier eine konkrete Kaufempfehlung zu geben fällt schwer. Zu groß und vielfältig ist das Angebot. Wer viel spielen möchte, sollte sich aber unbedingt vor Billigangeboten hüten. Sonst sind nach der ersten größeren Schlacht nicht nur die gegnerischen Raumschiffe zerstört, sondern gleich auch der Joystick!

Als äußerst robust erweisen sich Joysticks mit sogenannten Mikroschaltern zur internen Steuerung. Außerdem sollte man vor dem Kauf in jedem Fall einen "Grifftest" machen, d.h. prüfen, wie der Joystick in der Hand liegt und wie leicht oder schwer er sich bewegen läßt.

Joysticks lassen sich aber nicht nur bei Spielen sinnvoll einsetzen. Auch viele Grafikprogramme erlauben die Steuerung über

einen Joystick. Damit kann man dann zum Beispiel Freihandzeichnen oder Farben und Muster auswählen usw...

Der dritte Anwendungsbereich für einen Joystick ist GEOS. In GEOS, der grafischen Benutzeroberfläche des Commodore 64, haben Sie einen sogenannten Maus-Cursor (ähnlich dem normalen Text-Cursor), den Sie mit einem Joystick steuern können.

Als Alternative zum Joystick bietet sich eine sogenannte Maus an. Beide Zusatzgeräte werden am Joystick-Port angeschlossen. Worin liegen also die Unterschiede?

Der Joystick ist in erster Linie auf eine schnelle Reaktion auf Richtungsänderungen ausgelegt. Wenn Sie aber zum Beispiel einmal versuchen, den Mauszeiger von GEOS von der linken unteren in die rechte obere Bildschirmecke zu bewegen, werden Sie feststellen, daß das ganz schön lange dauert.

Eine Maus dagegen reagiert unmittelbar auf die Bewegung. So, wie Sie die Maus auf dem Tisch bewegen, bewegt sich auch der Mauszeiger am Bildschirm. Gleichzeitig erlaubt die Maus auch eine feinere Steuerung, etwa beim Freihandzeichnen in einem Grafikprogramm.

Allerdings gibt es für den Commodore 64 zwei verschiedene Maustypen. Wenn Sie sich eine Maus kaufen möchten, sollten Sie unbedingt darauf achten, daß es sich auch um eine sogenannte Proportional-Maus handelt. Andere Mäuse verhalten sich nämlich wie ein ganz normaler Joystick!

Die von Commodore angebotene Maus "C 1351" kostet etwa 90 Mark. Für einen guten Joystick muß man 50 Mark hinblättern.

Ein kleiner Tip für alle Maus-Interessierten: Wenn Ihnen zufällig auch ein Amiga-Rechner zur Verfügung stehen sollte, so können Sie dessen Maus problemlos am Commodore 64 anschließen! Die Amiga-Maus ist nämlich mit der Proportional-Maus C 1351 identisch.

1.6 Mit Software arbeiten

Software gibt es ganz allgemein in vielerlei "Form": auf Steckmodulen, auf Kassette, auf Diskette oder in gedruckter Form als sogenanntes Listing.

Programm-Listings müssen Sie zunächst einmal in den Rechner eintippen, bevor Sie die Programme nutzen können. Ein sehr mühseliges und vor allem fehlerträchtiges Unterfangen. Programm-Listings haben aber auch einen großen Vorteil: Sie können daraus lernen und Ihre Programmierkenntnisse verbessern - vorausgesetzt natürlich, Sie wollen überhaupt selbst programmieren. Aus diesem Grund finden Sie auch in diesem Buch eine Vielzahl von Listings in den beiden Programmiersprachen BASIC und Assembler. Zunächst wollen wir uns aber ausschließlich mit "gebrauchsfertiger" Software befassen.

1.6.1 Software auf Steckmodulen

Software auf Steckmodulen findet man mittlerweile nur noch relativ selten. Das liegt nicht zuletzt an den Preisen für die Hardware. Das Steckmodul allein (d.h. ohne die Software) kostet schon etwa 50 Mark, eine Diskette dagegen weniger als 2 Mark! Kaum ein Anwender ist heute noch bereit, diesen Mehrpreis zu bezahlen.

Steckmodule werden daher nur noch dann verwendet, wenn ein Programm derart groß und umfangreich ist, daß es mit einer Diskette nicht mehr sinnvoll eingesetzt werden kann. Ein Beispiel dafür ist das Programm "Pagefox", ein sogenanntes Desktop-Publishing-Programm, das ich Ihnen in Kapitel 2 vorstellen werde.

Die Arbeit mit einem Steckmodul gestaltet sich denkbar einfach. Das Steckmodul muß dazu nur in den Modul-Steckplatz auf der Rückseite des Commodore 64 ganz rechts eingesteckt werden.

Aber Vorsicht! Der Commodore 64 muß beim Ein- oder Ausstecken eines Moduls unbedingt ausgeschaltet sein. Ansonsten

riskieren Sie eine Beschädigung des Moduls und im schlimmsten Fall sogar des Rechners selbst. Beim Einschalten des Rechners prüft der Commodore 64 automatisch, ob sich im Modulschacht ein Modul befindet. Falls ja, übergibt er die Rechnerkontrolle an das Modulprogramm, das sich dann meist mit irgendeiner speziellen Einschaltmeldung bemerkbar macht. Wie Sie anschließend weiterverfahren müssen, erfahren Sie aus der Bedienungsanleitung zu dem Programm.

1.6.2 Software von Datasette und Floppy laden

Um Software von Kassette oder Diskette zum Laufen zu bringen, ist schon etwas mehr Aufwand erforderlich. Der ganze Vorgang gliedert sich in zwei wesentliche Schritte: Zunächst müssen Sie das Programm in den Speicher des Commodore 64 laden. Anschließend müssen Sie das Programm starten. Manche Programme starten sich nach dem Laden auch automatisch. Im folgenden müssen Sie nun einige Kommandos eingeben, die Ihnen im Augenblick vielleicht noch nicht sehr viel sagen. Das läßt sich leider nicht vermeiden. Bei allen Kommandos handelt es sich um Anweisungen in der Programmiersprache BASIC. Bitte achten Sie darauf, daß Sie jede Anweisung jeweils in eine leere Bildschirmzeile schreiben und zum Schluß die <Return>-Taste drücken.

Falls Sie einmal die Meldung SYNTAX ERROR bekommen sollten, haben Sie vermutlich etwas falsch eingetippt. Bitte überprüfen und verbessern Sie in diesem Fall Ihre Eingabe, und versuchen Sie es noch einmal. Der Commodore 64 ist etwas pingelig, was die Eingaben betrifft. Schon ein einzelnes vergessenes Komma oder Anführungszeichen führt in der Regel zu einer Fehlermeldung.

Mit der Datasette arbeiten

Die Datasette arbeitet grundsätzlich wie ein Kassettenrekorder. Nachdem Sie die Programmkassette in die Datasette eingelegt und gegebenenfalls bis zum Anfang zurückgespult haben, kann es losgehen. Bitte geben Sie ein:

```
LOAD
```

Der Commodore 64 antwortet Ihnen daraufhin mit:

```
PRESS PLAY ON TAPE
```

Sie sollen also die <Play>-Taste der Datasette drücken. Bitte tun Sie dies, aber erschrecken Sie nicht! Daß der Bildschirm nun die Rahmenfarbe annimmt und der Text vom Bildschirm verschwindet, ist völlig normal und hat technische Gründe. Immer wenn der Commodore 64 etwas von der Datasette lädt, schaltet er, um Zeit zu sparen, den Bildschirm ab. Das Laden von Datasette dauert aber auch so noch lange genug, wie Sie schon bald feststellen werden.

Der Commodore 64 sucht nun das erste auf der Kassette gespeicherte Programm. Hat er es gefunden, meldet er sich wieder am Bildschirm:

```
FOUND PROGRAMMNAME
```

Hinter FOUND steht der Name des gefundenen Programms. Jetzt müssen Sie nur noch die <Commodore>-Taste links unten auf der Tastatur drücken und das Programm wird geladen.

Falls sich das Programm nicht selbst startet, meldet sich der Rechner nach Beendigung des Ladevorgangs wieder mit einer READY.-Meldung. Das Programm befindet sich nun im Speicher des Commodore 64 und muß nur noch gestartet werden. Dazu geben Sie ein:

```
RUN
```

Der gesamte Vorgang läßt sich auch abkürzen: Drücken Sie die <Shift>- und die <Run/Stop>-Taste zusammen. Der Commodore 64 fordert Sie dann wieder auf, die <Play>-Taste der Datasette zu drücken, lädt das gefundene Programm und startet es automatisch.

Es kann auch sein, daß das Programm eine spezielle Startanweisung benötigt. Diese finden Sie dann in der Bedienungsanleitung zu dem Programm, meist auf den ersten paar Seiten.

Falls sich auf der Kassette mehrere Programme befinden und Sie ein bestimmtes Programm laden möchten, müssen Sie hinter LOAD den Namen des zu ladenden Programms angeben:

```
LOAD "PROGRAMMNAME"
```

Bitte beachten Sie, daß der Programmname in Anführungszeichen stehen muß! Die weitere Vorgehensweise ist dieselbe wie gerade eben beschrieben. Wenn sich das Programm weiter hinten auf der Kassette befindet, kann die Suche aber sehr lange dauern. Besser ist es da, das Band ungefähr bis zu der Stelle vorzuspulen, an der man das Programm vermutet.

Nicht mehr lange und Sie werden sicher Ihre ersten eigenen BASIC-Programme schreiben. Um diese dann dauerhaft zu sichern, müssen Sie sie auf Kassette speichern. Dazu benötigen Sie das SAVE-Kommando:

```
SAVE "PROGRAMMNAME"
```

Der Name Ihres Programms darf bis zu 16 Zeichen lang sein. Bei längeren Namen werden nur die ersten 16 Zeichen berücksichtigt, Sie erhalten also bei zu langen Namen keine Fehlermeldung.

Nachdem Sie die <Return>-Taste gedrückt haben, erhalten Sie vom Commodore 64 die Aufforderung:

```
PRESS PLAY AND RECORD ON TAPE
```

Sie sollen also die <Play>- und die <Record>-Taste der Datasette gleichzeitig drücken, wie man es auch beim Aufnehmen bei einem "normalen" Kassettenrekorder gewohnt ist. Anschließend wird das Programm auf Kassette gespeichert. Auch dabei schaltet der Commodore 64 wieder den Bildschirm ab.

Mit LOAD "PROGRAMMNAME" können Sie Ihr Programm jetzt jederzeit wieder in den Rechner laden. Diese Informationen zur Datasette sollen fürs erste genügen. Viele weitere Informationen zur Handhabung der Datasette finden Sie in Kapitel 10.

Mit der Floppy arbeiten

Die Arbeit mit der Floppy 1541 gestaltet sich etwas anders als mit der Datasette. Zunächst einige Worte zum Speichermedium, den Disketten. Disketten sind sehr empfindlich! Man sollte sie daher sehr pfleglich behandeln, insbesondere, was die Aufbewahrung betrifft. Keinesfalls dürfen Disketten in der Nähe von Magnetfeldern gelagert werden. Auch gegen Hitze und Feuchtigkeit reagieren sie sehr allergisch.

Leere Disketten, auf denen Sie selbstgeschriebene Programme oder Daten speichern können, erhalten Sie im Fachhandel im 10er-Pack und in Preislagen von knapp 10 Mark bis über 40 Mark. Die billigsten Disketten mit der Bezeichnung "1S/1D" können Sie zwar prinzipiell verwenden, im Hinblick auf die Datensicherheit empfehlen sich aber Disketten der Qualität "2S/1D" oder "2S/2D", die für etwa 20 bis 25 Mark zu haben sind (wohlgemerkt 10 Stück). Das ist zwar mehr als doppelt soviel wie bei den Billig-Disketten. Spätestens nach dem ersten Programm- oder Datenverlust werden Sie aber feststellen, daß Billigangebote auch ihre Nachteile haben.

Ein Vorgang, mit dem sich jeder Einsteiger etwas schwertut, ist das Einlegen einer Diskette in das Laufwerk:

- Öffnen Sie zunächst das Laufwerk. Die Art des Verschlusses ist etwas unterschiedlich, je nach dem Alter Ihres Floppy-Modells. Bei den älteren Modellen mit einem länglichen Verschluß müssen Sie diesen leicht nach hinten drücken. Alles weitere besorgt dann eine im Gehäuse eingebaute Feder. Die neueren Floppy-Modelle verfügen über einen sogenannten Knebelverschluß, den man ähnlich einem Schalter nach oben drehen muß.
- Nun kommt das Wichtigste, das "Ausrichten" der Diskette. Nehmen Sie einmal eine Diskette zur Hand, am besten die

GEOS-Diskette, die Sie auf jeden Fall haben müßten, und sehen sie sich etwas genauer an. Bitte fassen Sie die Diskette am Etikett an, und halten Sie sie vom Körper weg. Wenn Sie jetzt auf die Diskette sehen, müßte sich das Etikett rechts vorne befinden (oder sich über die gesamte Vorderseite erstrecken).

Auf der linken Seite der Diskette sehen Sie eine rechteckige Einkerbung. Das ist die sogenannte Schreibschutzkerbe. Wenn Sie diese Kerbe zukleben (passende Aufkleber finden Sie in jeder Packung mit leeren Disketten), kann auf die Diskette nichts mehr geschrieben werden. Das ist sehr wichtig, wenn Sie wichtige Daten vor dem versehentlichen Überschreiben oder Löschen schützen wollen.

Manche Disketten mit gekauften Programmen haben auch auf der rechten Seite eine Einkerbung, so z.B. die GEOS-Disketten. In diesem Fall sind die Disketten auch auf der Rückseite beschrieben. Leere Disketten, die über nur eine Schreibschutzkerbe verfügen, können mit einer Schere leicht mit einer zusätzlichen Kerbe versehen werden. Dadurch erreichen Sie die doppelte Speicherkapazität. Allerdings ist die Floppy 1541 an sich nur auf die Arbeit mit einseitig beschreibbaren Disketten eingerichtet. In seltenen Fällen kann es daher bei zweiseitigen Disketten zu einem Datenverlust kommen. Besser also, Sie verwenden Ihre Disketten nur einseitig.

Auf der dem Etikett gegenüberliegenden Seite sehen Sie einen größeren Einschnitt in die Diskettenhülle. Dort werden die Daten gelesen bzw. geschrieben. Bitte achten Sie unbedingt darauf, daß Sie diese Stellen nie berühren. Dadurch könnten unter Umständen nicht nur Ihre Daten verloren gehen, auch die Floppy selbst könnte Schaden nehmen! Der Schreib-/Lesekopf der Floppy reagiert schon auf kleinste Schmutzpartikel (Staub, Zigarettenasche, Fingerabdrücke) sehr empfindlich. Disketten gehören daher, sofern sie sich nicht im Laufwerk befinden, immer in ihre Hülle!

So, wie Sie die Diskette gerade in der Hand halten, wird Sie auch in das Laufwerk eingeführt. Je nachdem, ob Sie die Vor-

der- oder Rückseite verwenden wollen, entweder mit dem Etikett nach oben oder nach unten. Bitte schieben Sie die Diskette langsam in die Floppy, bis Sie einen leichten Widerstand spüren.

- Zum Schluß müssen Sie nur noch das Laufwerk schließen. Jetzt ist die Diskette "betriebsbereit", d.h., sie kann gelesen oder beschrieben werden.

Um die Diskette wieder aus dem Laufwerk zu bekommen, verfahren Sie analog. Sobald Sie den Laufwerksverschluß öffnen, wird die Diskette automatisch "ausgeworfen", d.h., sie wird durch eine Feder etwa um die Hälfte ihrer Länge aus dem Laufwerk geschoben, so daß Sie sie leicht entnehmen können. Und dann sofort ab in die Papierhülle damit!

Nehmen wir jetzt einmal an, Sie haben eine Diskette ins Laufwerk eingelegt. Um nun ein Programm zu laden, geben Sie ein:

```
LOAD "PROGRAMMNAME",8
```

Bitte vergessen Sie nicht, die <Return>-Taste zu drücken! Als Programmnamen geben Sie den Namen an, den Sie in der Bedienungsanleitung zu Ihrem Programm finden. Viele kommerzielle Programme verwenden anstelle eines langen Namens einfach einen Stern (*). Es kann auch sein, daß Sie an die (8) (dabei handelt es sich übrigens um die sogenannte Geräteadresse der Floppy) noch ein (,1) anhängen müssen. In diesem Fall startet sich das Programm dann automatisch.

Wenn Sie alles richtig gemacht haben, müßten nun folgende Bildschirmmeldungen erscheinen:

```
SEARCHING FOR PROGRAMMNAME
LOADING
```

Wenn sich anschließend der Rechner wieder mit einer READY-Meldung meldet, müssen Sie das Programm noch starten. Dazu geben Sie ein:

```
RUN
```

Zum Abspeichern von selbstgeschriebenen oder abgetippten BASIC-Programmen benötigen Sie das SAVE-Kommando:

```
SAVE "PROGRAMMNAME",8
```

Der Programmname darf maximal 16 Zeichen umfassen, überzählige Zeichen werden vom Rechner automatisch "abgeschnitten"; eine Fehlermeldung erhalten Sie nicht.

Bitte beachten Sie, daß sich auf einer Diskette keine zwei Programme oder Dateien mit demselben Namen befinden dürfen. Das hätte ja auch keinen Sinn, denn gerade der Name dient ja zur Unterscheidung der einzelnen Programme.

Wenn Sie versuchen, ein Programm unter einem Namen abzuspeichern, der auf der eingelegten Diskette bereits existiert, reagiert die Floppy mit einer Fehlermeldung. Daß etwas nicht stimmt, erkennen Sie auch daran, daß die rote LED an der Vorderseite der Floppy hektisch blinkt. In diesem Fall ändern Sie bitte den Namen des Programms und versuchen noch einmal, es abzuspeichern. Klappt es nun immer noch nicht, so liegt vermutlich ein anderer Fehler vor. Es würde hier zu weit führen, Ihnen alle möglichen Fehler und vor allem deren Abhilfe zu erklären, da dazu zumindest Grundkenntnisse in der Programmiersprache BASIC erforderlich sind, die Sie ja im Moment noch nicht haben.

Auf zwei wichtige Vorgänge möchte ich aber trotzdem kurz eingehen: das Formatieren einer Diskette und das Löschen von Programmen.

Daß es mit dem Speichern eines Programms nicht klappt, kann einen ganz einfachen Grund haben. Vielleicht haben Sie eine neu gekaufte Diskette ins Laufwerk eingelegt, und diese ist nun natürlich noch nicht formatiert. Was versteht man nun darunter schon wieder?

Die Disketten, die Sie für Ihre Floppy 1541 kaufen, lassen sich auch für verschiedene andere Computer bzw. deren Laufwerke verwenden. Jedes Laufwerk hat aber sein spezielles Aufzeichnungsformat. Damit bezeichnet man die Art und Weise, wie die Daten vom Laufwerk auf der Diskette gespeichert werden. Um sich nun später auf der Diskette "zurechtzufinden", muß das Laufwerk zunächst eine Reihe von Markierungen auf die Diskette schreiben. Diesen Vorgang bezeichnet man als "Formatieren" der Diskette.

Aber Vorsicht: Durch das Formatieren einer Diskette gehen alle auf der Diskette eventuell gespeicherten Daten unwiederbringlich verloren! Bitte achten Sie also darauf, daß Sie nicht eine Diskette mit wichtigen Programmen oder Daten aus Versehen neu formatieren. Um eine Diskette zu formatieren, geben Sie die folgende Zeile ein:

```
OPEN 1,8,15,"N:DISKETTENNAME,ID":CLOSE 1
```

Als Diskettenname dürfen Sie einen beliebigen, bis zu 16 Zeichen langen Text verwenden. Die ID dient der Floppy zur Unterscheidung der einzelnen Disketten. Jede Ihrer Disketten sollte daher über eine andere ID verfügen. Als ID sind beliebige zweistellige Buchstaben- oder Ziffernkombinationen erlaubt, beispielsweise "AB", "A1", "01" usw... Am einfachsten ist es, Sie "numerieren" sich Ihre Disketten von "01" bis "99" durch. Wenn es dann irgendwann mehr als 100 Disketten sind, können Sie ja mit dem Alphabet weitermachen.

Nachdem Sie die obige Zeile eingetippt und die <Return>-Taste gedrückt haben, beginnt die Floppy mit dem Formatieren. Bitte erschrecken Sie nicht! Das laute Rattern am Anfang ist völlig normal. Dabei wird nur der Schreib-/Lesekopf der Floppy justiert. Der gesamte Formatiervorgang dauert etwa 90 Sekunden. Während dieser Zeit ist auch der Commodore 64 nicht "ansprechbar". Die READY-Meldung erscheint erst, nachdem auch die rote LED der Floppy erloschen ist.

Die rote LED der Floppy leuchtet übrigens immer dann, wenn sich die Floppy in irgendeiner Weise mit der eingelegten Dis-

kette befaßt. Solange die LED leuchtet, sollte man die eingelegte Diskette daher keinesfalls aus dem Laufwerk entfernen, das könnte fatale Folgen haben!

Wenn man längere Zeit an einem BASIC-Programm arbeitet, kommt man nicht umhin, das Programm von Zeit zu Zeit abzuspeichern. Dazu möchte man dann natürlich nicht immer wieder einen neuen Namen verwenden. Sonst wäre die Diskette schon bald mit vielleicht 10 oder 15 verschiedenen Programmversionen überfüllt. Das Programm einfach unter dem alten Namen abzuspeichern geht aber, wie Sie ja inzwischen wissen, auch nicht. Die Lösung: Das alte Programm muß auf der Diskette gelöscht werden. Dazu geben Sie die folgende Zeile ein:

```
OPEN 1,8,15,"S:PROGRAMMNAME":CLOSE 1
```

Das "S" vor dem Programmnamen steht für das englische "Scratch", also "Löschen". Anschließend können Sie das Programm mit dem üblichen SAVE "PROGRAMMNAME",8 abspeichern. Damit möchte ich es fürs erste bewenden lassen. Viele weitere Informationen zur Arbeit mit der Floppy finden Sie in Kapitel 5.

Egal, ob Sie mit einer Datasette oder einer Floppy arbeiten, Sie sollten sich nach Möglichkeit immer eine Sicherheitskopie Ihrer Programme erstellen. Eine Kassette oder Diskette ist schnell beschädigt! Wenn Sie dann über keine Kopie verfügen, ist das Programm verloren.

Voraussetzung für das Kopieren ist allerdings, daß die betreffenden Programme über keinen Kopierschutz gegen das Raupkopieren verfügen. Falls dies der Fall sein sollte, besteht meistens die Möglichkeit, beim Hersteller des Programms für einen kleinen Unkostenbetrag eine Ersatzdiskette anzufordern.

Das Kopieren selbst geht ganz einfach. Eine Kassette kopieren Sie wie eine normale Musikkassette mit Hilfe zweier Kassettenrekorder oder Ihrer Stereoanlage. Zum Kopieren einer Diskette benötigen Sie ein Kopierprogramm, am besten ein sogenanntes Backup-Programm, das in der Lage ist, eine komplette Diskette

zu kopieren. Wenn Sie es vielleicht auch noch nicht wissen, solch ein Programm haben Sie schon! Der Benutzeroberfläche GEOS ist nämlich ein Backup-Programm beigefügt, das an sich zum Kopieren der GEOS-Disketten gedacht ist, sich aber auch für beliebige Disketten verwenden läßt. Die genaue Vorgehensweise erfahren Sie im nächsten Abschnitt. Mit GEOS kann man übrigens auch sehr komfortabel einzelne Programme kopieren. Und nicht nur das: GEOS bietet einiges, das einem die Verwaltung seiner Disketten wesentlich vereinfacht. Grund genug, sich GEOS einmal genauer anzuschauen.

1.6.3 Selbst programmieren

Die vielleicht interessanteste Tätigkeit im Zusammenhang mit Computern ist das Programmieren. Doch, was versteht man eigentlich unter "programmieren"?

Zum Programmieren benötigen Sie eine sogenannte Programmiersprache. Ähnlich wie in einer "natürlichen" Sprache teilen Sie dem Computer in dieser Sprache mit, was er machen soll. Auf dem Commodore 64 stehen Ihnen zwei Programmiersprachen zur Verfügung: BASIC und Assembler.

BASIC steht für "Beginners All Symbolic Instruction Code", was soviel heißt wie "symbolische Sprache für Einsteiger". BASIC ist die ideale Einsteigersprache. Schon mit sehr wenigen Sprachelementen lassen sich komplette Programme schreiben. Außerdem ist BASIC eine sehr anschauliche, anwendungsorientierte Sprache.

Die Sprache "Assembler" orientiert sich stark am internen Aufbau des Commodore 64 und ist daher wesentlich schwerer zu verstehen als BASIC. Bevor man sich mit Assembler befaßt, sollte man sich unbedingt genau mit BASIC auskennen. Bleiben wir also einmal bei BASIC. Wenn Sie Ihren Commodore 64 eingeschaltet haben und der Cursor am Bildschirm blinkt, ist der Rechner zum Programmieren in BASIC bereit.

Wie sieht nun ein BASIC-Programm aus?

Nehmen wir dazu ein ganz einfaches Beispiel. Sie wollen zwei Zahlen addieren. Geben Sie nun einmal folgendes ein:

```
PRINT 10+2
```

Bitte vergessen Sie nicht, die <Return>-Taste zu drücken! Nachdem Sie dies getan haben, müßte der Bildschirm wie folgt ausschauen:

```
PRINT 10+2
12

READY.
■
```

Der Commodore 64 hat die beiden Zahlen 10 und 2 addiert und das Ergebnis auf dem Bildschirm ausgegeben. Die Anweisung PRINT signalisiert dem Rechner, das er etwas drucken soll: in unserem Fall das Ergebnis der Addition.

Hinter PRINT dürfen beliebige Ausdrücke stehen. Probieren Sie doch einmal:

```
PRINT "HALLO"
```

Die beiden Anführungszeichen zeigen dem Rechner an, das er einen Text ausgeben soll.

Das Ganze ist zwar schon recht nett, aber wenig dauerhaft. Wenn Sie die <Cursor down>-Taste lange genug drücken, verschwindet die PRINT-Anweisung für immer vom Bildschirm. Wie sichert man die PRINT-Anweisung dauerhaft? Dazu stellen Sie dem PRINT eine Nummer voran, zum Beispiel 100:

```
100 PRINT "HALLO"
```

Wenn Sie das eingegeben haben und dann die <Return>-Taste drücken, passiert zunächst gar nichts. Nur der blinkende Cursor wandert an den Anfang der nächsten Bildschirmzeile.

Geben Sie nun aber einmal folgendes ein:

```
LIST
```

Und siehe da! Die gerade eingegebene Zeile erscheint noch einmal auf dem Bildschirm. Der Commodore 64 hat diese Zeile in seinem internen Speicher abgelegt, von wo sie mit LIST jederzeit abgerufen werden kann. Diese eine Zeile stellt im Grunde genommen schon ein vollwertiges, wenn auch sehr simples BASIC-Programm dar. Geben Sie nun einmal ein:

```
RUN
```

Der Bildschirm müßte anschließend so aussehen:

```
100 PRINT "HALLO"

RUN
HALLO

READY.
■
```

Durch die RUN-Anweisung haben Sie das BASIC-Programm gestartet. Nachdem der Commodore 64 das Programm abgearbeitet hat, kehrt er in den Direktmodus zurück, was er Ihnen durch die READY-Meldung anzeigt.

Doch kommen wir auf unser Eingangsproblem zurück. Wir wollten ja zwei Zahlen addieren. Die Addition mit Hilfe der PRINT-Anweisung durchzuführen ist zwar nicht schlecht, viel bequemer wäre es aber doch, wenn der Rechner nach den zu addierenden Zahlen fragen und anschließend das Ergebnis ausgeben würde.

Dazu benötigen wir eine weitere Anweisung: INPUT. INPUT (englisch für Eingabe) signalisiert dem Rechner, das er vom Anwender etwas erfragen soll. Bitte geben Sie einmal folgendes kleine Programm ein:

```
100 INPUT "BITTE GEBEN SIE DIE ERSTE ZAHL EIN:";Z1
110 INPUT "BITTE GEBEN SIE DIE ZWEITE ZAHL EIN:";Z2
120 EG=Z1+Z2
130 PRINT "ERGEBNIS: ";EG
```

Wenn Sie dieses Programm mit RUN starten, ergibt sich folgender Dialog:

```
BITTE GEBEN SIE DIE ERSTE ZAHL EIN: ? 1000
BITTE GEBEN SIE DIE ZWEITE ZAHL EIN: ? 267
ERGEBNIS: 1267
```

Die beiden Zahlen hinter den Fragezeichen müssen Sie selbst eingeben und anschließend die <Return>-Taste drücken. Mit diesem Programm können Sie zwei beliebige Zahlen sehr komfortabel addieren lassen. Damit wollen wir es fürs erste bewenden lassen. Wie Sie gesehen haben, ist das Ganze doch gar nicht so schwer. Wenn ich Sie jetzt auf den Geschmack gebracht haben sollte und Sie nun darauf brennen, Ihre ersten eigenen Programme zu schreiben, so können Sie gleich zu Kapitel 3 weiterblättern.

1.7 Mit GEOS arbeiten

GEOS ist die erste "professionelle" Software, mit der Sie arbeiten werden. In diesem Zusammenhang kann ich Ihnen gleich noch einige Fachbegriffe und Verfahrensweisen erklären:

- GEOS läßt sich leider nur mit einem Diskettenlaufwerk einsetzen, Datasetten-Besitzer können mit GEOS nichts anfangen.

GEOS steht für "Graphic Environment Operating System", also ein Betriebssystem mit grafischer Bedienungsumgebung. Im Gegensatz zum "normalen" Betriebssystem des Commodore 64, mit dem Sie schon etwas Bekanntschaft gemacht haben, zum Beispiel, als Sie eine Diskette formatierten, arbeitet GEOS fast vollständig grafisch und läßt sich intuitiv bedienen. Was das im einzelnen genau bedeutet, werden Sie gleich sehen.

Bekanntlich gibt es ja nichts, was man nicht noch verbessern könnte. Das gilt insbesondere für Software. Auch wenn ein Programm noch so sorgfältig getestet wurde, bevor es in den Verkauf geht, irgendein Fehler findet sich später garantiert immer. Und natürlich haben auch die Programmierer immer wieder neue Ideen, wie sie ihre Programme noch leistungsfähiger machen können. Aus diesem Grund gibt es von Programmen in der Regel mehrere Versionen, so auch bei GEOS.

Die GEOS-Version, die Ihrem Commodore 64 beiliegt, trägt die Versionsnummer 1.2. Mittlerweile gibt es aber schon die Version 2.0, die gegenüber der Version 1.2 zahlreiche Verbesserungen aufweist. Das zeigt sich schon am Lieferumfang: GEOS 2.0 wird auf vier Disketten und mit einem über 300 Seiten umfassenden Handbuch ausgeliefert. Ein zusätzlicher Vorteil: GEOS 2.0 ist im Gegesatz zur Version 1.2 komplett eingedeutscht, empfängt Sie also am Bildschirm mit deutschen Meldungen und Bezeichnungen anstatt mit englischen.

Als Besitzer der Version 1.2 haben Sie die Möglichkeit, ein Update auf die Version 2.0 zu machen. Was ist nun das schon wieder, ein Update? Da wohl kaum ein Anwender bereit ist, eine neue Programmversion jedesmal neu zu kaufen, bieten die meisten Software-Hersteller ihren alten Kunden neue Programmversionen zu Vorzugskonditionen an. Solch ein "Update" gibt es meist deutlich billiger als das Programm für Neukunden. Das Update auf GEOS 2.0 kostet beispielsweise nur 49 Mark, während ein Neukunde 89 Mark bezahlen muß.

Nun stellt sich natürlich die Frage, ob es sich überhaupt lohnt, die 49 Mark für GEOS 2.0 auszugeben. Um Ihnen die Entscheidung zu erleichtern, werde ich am Ende dieses Abschnitts auf die Erweiterungen und Neuheiten in GEOS 2.0 besonders eingehen. Bevor wir uns GEOS genauer anschauen, sollten Sie sich zuallererst eine Sicherheitskopie sowie eine Arbeitsdiskette der GEOS-Originaldiskette erstellen. Dazu gibt es auf der GEOS-Diskette ein spezielles Kopierprogramm, das Sie, wie schon weiter oben erwähnt, zum Kopieren beliebiger Disketten verwenden können.

Zunächst müssen Sie das Kopierprogramm laden und starten. Bitte legen Sie dazu die GEOS-Diskette ins Laufwerk, und geben Sie die folgenden zwei Zeilen ein:

```
LOAD "BACKUP",8
RUN
```

Nachdem das Programm gestartet wurde, müßte auf dem Bildschirm folgender Text erscheinen:

```
DISK BACKUP/RESTORE UTILITY
INSERT DESTINATION DISK TO BE FORMATED
AND ENTER 'F' TO FORMAT, OR 'Q' TO QUIT (F/Q)
```

Legen Sie eine leere Diskette ins Laufwerk, und drücken Sie anschließend die Taste <F> sowie die <Return>-Taste. Die Diskette wird nun formatiert, was Ihnen der Commodore 64 auch am Bildschirm mitteilt:

```
FORMATTING DESTINATION DISK
```

Anschließend fordert Sie der Rechner auf, wieder die Originaldiskette einzulegen, damit der Kopiervorgang gestartet werden kann:

```
INSERT SOURCE DISK AND ENTER 'C' TO
COPY (C)
```

Drücken Sie also die Taste <C> und anschließend die <Return>-Taste. Sie erhalten dann auf dem Bildschirm die Meldung:

```
READING SOURCE DISK
```

Nach einiger Zeit fordert Sie der Rechner dazu auf, die gerade formatierte Diskette einzulegen:

```
PLEASE INSERT DESTINATION DISK
```

Nachdem sie das getan haben, erscheint die Meldung:

```
WRITING DESTINATION DISK
```

Die von der Originaldiskette (Quelldiskette oder Source Disk) gelesenen Daten werden nun auf die Sicherheitskopie (Zieldiskette oder Destination Disk) geschrieben. Da der Inhalt einer Diskette nicht komplett in den Speicher des Commodore 64 paßt, wiederholt sich dieser Vorgang noch zweimal, d.h., Sie werden noch zweimal aufgefordert, die "Source Disk" und anschließend wieder die "Destination Disk" einzulegen. Die Bezeichnungen "Source Disk" und "Destination Disk" für die Quell- bzw. die Zieldiskette werden auch von anderen Programmen, sofern diese nicht eingedeutscht sind, sehr häufig verwendet. Sie sollten sich diese Bezeichnungen merken.

Nachdem der Kopiervorgang beendet ist, erscheint die Meldung:

```
BACKUP COMPLETE!
INSERT GEOS BOOT DISK
AND PRESS RESTORE
```

Legen Sie also wieder die GEOS-Originaldiskette ins Laufwerk, und drücken Sie die <Restore>-Taste (diese befindet sich direkt über der <Return>-Taste auf der Tastatur).

GEOS wird nun gebootet! Wieder so ein englischer Fachausdruck. "Booten" heißt nichts anderes, als daß das Programm in den Rechnerspeicher geladen und gestartet wird.

Leider verfügt GEOS über einen etwas eigenwilligen Kopierschutz, der den Ladevorgang mitunter auch bei einer ordnungsgemäß erworbenen Originaldiskette abbrechen läßt. In diesem Fall erhalten Sie nach einigen Sekunden wieder die gewohnte

Einschaltmeldung des Commodore 64. Versuchen Sie nun, GEOS noch einmal zu starten, indem Sie eingeben:

```
LOAD "GEOS",8,1
```

Sollte es auch nach mehreren Versuchen nicht klappen, hilft nur noch der Gang zum Fachhändler. Entweder die GEOS-Diskette ist defekt oder Ihr Laufwerk ist verstellt.

Hat alles geklappt, so müßten Sie jetzt eine doch recht ansprechende Grafik auf dem Bildschirm haben. Das sieht doch schon ganz anders aus als die normale Einschaltmeldung des Commodore 64!

Wenn Sie jetzt allerdings versuchen, etwas über die Tastatur einzutippen, werden Sie feststellen, daß Sie keine Reaktion bekommen. Nehmen Sie nun aber einmal den Joystick zur Hand (der Joystick muß sich im Control Port 1 befinden), und bewegen Sie ihn. Wie Sie sehen, bewegt sich der kleine, schräggestellte Pfeil am Bildschirm entsprechend der Joystick-Bewegung. Diesen Pfeil bezeichnet man als Maus-Cursor oder Mauszeiger. "Maus" deshalb, weil man ihm am besten mit einer sogenannten Proportional-Maus steuert (die aber deutlich teurer als ein Joystick ist). Für den Moment reicht aber ein einfacher Joystick völlig aus.

Der Mauszeiger läßt sich am ehesten mit dem gewohnten Cursor vergleichen. Wie Ihnen ja auch der Cursor anzeigt, an welcher Bildschirmposition Sie sich gerade befinden, so markiert Ihnen auch der Mauszeiger die Bildschirmstelle, an der Ihre "Eingaben" etwas bewirken.

Das, was Sie jetzt gerade am Bildschirm sehen, bezeichnet man als den Desktop von GEOS, was man am ehesten mit "Schreibtisch" übersetzen könnte. Links oben sehen Sie eine sogenannte Menüzeile, über die Ihnen die verschiedenen Programmfunktionen angeboten werden. In der rechten unteren Ecke "steht" ein Papierkorb, mit dem man Programme löschen kann.

In der großen, hell unterlegten Fläche finden Sie verschiedene Informationen zu der im Laufwerk eingelegten Diskette. In der obersten Zeile steht der Name der Diskette, in der Zeile darunter, wie viele Dateien sich auf der Diskette befinden, wieviel Speicherplatz belegt ist und wieviel Speicher noch frei ist.

Auf der großen Fläche darunter sehen Sie die einzelnen Programme, dargestellt durch ihren Namen, und eine ihre Funktion verdeutlichende Kleingrafik, ein sogenanntes Piktogramm oder Icon. Wegen der Icons können auf der Fläche maximal acht Dateien auf einmal untergebracht werden. Da auf einer Diskette natürlich wesentlich mehr Dateien Platz finden, gibt es mehrere "Seiten", die Sie mit Hilfe der Zifferntasten auf der Tastatur "durchblättern" können. Im Augenblick befinden wir uns auf Seite 1.

Das Programm rechts unten wird Ihnen schon bekannt vorkommen, es ist unser Kopierprogramm BACKUP. Das kommt uns gerade recht, denn wir benötigen ja noch eine zweite Kopie der GEOS-Originaldiskette, eine Arbeitsdiskette. Nun lernen Sie auch gleich, wie man unter GEOS ein Programm startet:

Bitte bewegen Sie den Mauszeiger (mit Hilfe des Joysticks oder einer Maus) etwa in die Mitte des BACKUP-Icons. Nun drücken Sie - kurz hintereinander - zweimal den Feuerknopf des Joysticks (oder die linke Taste der Maus). Das ist schon alles! Das BACKUP-Icon wird jetzt invertiert dargestellt, und am Arbeitsgeräusch der Floppy erkennen Sie, daß das Programm geladen wird. Nach einiger Zeit sehen Sie die bekannte Einschaltmeldung des Kopierprogramms am Bildschirm. Nun verfahren Sie analog zur Erstellung der Sicherheitskopie. Nachdem der Kopiervorgang beendet ist, müßten Sie über drei Disketten verfügen:

- GEOS-Originaldiskette. Diese werden Sie künftig nur noch zum Booten von GEOS benötigen.
- Sicherheitskopie. Diese bewahren Sie bitte an einem sicheren Ort auf. Sollte Ihre Originaldiskette einmal beschädigt werden, so können Sie sie evtl. mit Hilfe der Sicherheitskopie "reparieren".

- Arbeitsdiskette. Wie der Name schon sagt, werden Sie mit dieser Diskette in Zukunft arbeiten.

Konkret sieht das so aus: Zunächst booten Sie GEOS mit der Originaldiskette. (Wegen des Kopierschutzes geht das leider nicht von der Arbeitsdiskette.) Anschließend nehmen Sie die Originaldiskette aus dem Laufwerk und legen anstelle dieser die Arbeitsdiskette ein.

Tun Sie das gleich einmal! Nach Beendigung des Kopiervorgangs müßten Sie sich ja wieder im GEOS-Desktop befinden.

Wenn Sie die Diskette im Laufwerk wechseln, so müssen Sie das GEOS auch mitteilen. Bewegen Sie dazu den Mauszeiger auf das Wort "Disk" in der Menüzeile links oben. Wichtig ist, daß die Pfeilspitze auf das Wort zeigt. Drücken Sie nun den Feuerknopf des Joysticks (oder die linke Taste Ihrer Maus). Daraufhin "rollt" ein Untermenü herunter und der Mauszeiger befindet sich auf dem ersten Menüpunkt "Open". Bitte drücken Sie jetzt noch *einmal den Joystick-Knopf.*

Daraufhin verschwindet das Untermenü wieder, die Floppy arbeitet einige Zeit und anschließend wird der Bildschirminhalt neu aufgebaut. Was Sie gerade getan haben, bezeichnet man als das Öffnen einer Diskette.

Wo wir schon bei den Fachbegriffen sind: Das Bewegen des Mauszeigers an eine bestimmte Stelle (ein Icon oder ein Menüpunkt) bezeichnet man auch als "Anwählen", das Drücken des Joystick-Knopfes nennt man "Anklicken". Um nun ein Programm zu starten, muß man es "Doppelklicken", also zweimal kurz hintereinander anklicken, wie Sie es ja schon vorhin beim BACKUP-Programm getan haben.

Diese wenigen Bedienungselemente reichen schon aus, um mit GEOS zu arbeiten! Während Sie beim normalen Betriebssystem immer irgend etwas eintippen müssen (ganz kompliziert wurde es beispielsweise beim Formatieren einer Diskette), präsentiert Ihnen GEOS (und auch *alle Anwendungsprogramme, die unter* GEOS arbeiten) seine Programmfunktionen in Form von Icons

und Menüs, die Sie nur noch auswählen (also mit dem Mauszeiger daraufzeigen) und anklicken müssen.

Der Vorteil liegt klar auf der Hand: Man muß nicht erst komplizierte Befehle auswendig lernen, sondern kann sofort mit einem Programm loslegen. Da ja Zeit (hier: Einarbeitungszeit) bekanntlich Geld ist, haben solche Benutzeroberflächen, wie GEOS, auch im professionellen Bereich weite Verbreitung gefunden. Bekannte Namen sind beispielsweise "GEM" und "Windows" für Personalcomputer unter dem Betriebssystem MS-DOS oder "X-Windows" für das Betriebssystem Unix. Diese Programme bieten natürlich entsprechend der höheren Leistungsfähigkeit der Rechner, auf denen sie eingesetzt werden, einiges mehr als GEOS. Die grundsätzliche Bedienungsweise ist aber dieselbe!

Zurück zu unserer GEOS-Arbeitsdiskette: Im Moment ist diese mit der Originaldiskette noch vollkommen identisch; das soll sich gleich ändern.

Zunächst benötigen wir einen anderen Namen, damit GEOS die Arbeitsdiskette von der Originaldiskette unterscheiden kann. Klicken Sie dazu bitte das Menü "Disk" und anschließend den Menüpunkt "Rename" an. GEOS baut nun ein sogenanntes Fenster auf dem Bildschirm auf und fordert Sie mit PLEASE ENTER NEW DISK NAME zur Eingabe des neuen Diskettennamens auf. Eine Zeile darunter steht der alte Name der Diskette, den Sie, wie gewohnt, mit der <Inst/Del>-Taste löschen können. Der neue Name darf bis zu 16 Zeichen lang sein, beispielsweise "GEOS-Arbeitsdisk".

Wie Sie schon bald feststellen werden, ist der freie Speicherplatz auf der Diskette unter GEOS chronisch knapp. Die GEOS-Arbeitsdiskette ist sogar "randvoll". Höchste Zeit also, etwas Platz zu schaffen. Die drei Programme "GEOS", "GEOS BOOT" und "GEOS KERNAL" werden nur zum Booten von GEOS benötigt. Da Sie mit der Arbeitsdiskette ja ohnehin nicht booten können, dürfen Sie diese Programme löschen. Wie wird nun ein Programm unter GEOS gelöscht?

Um die drei Programme überhaupt löschen zu können, müssen wir zunächst deren "Löschschutz" entfernen. Bitte klicken Sie dazu das Icon eines Programms an, beispielsweise das GEOS-Icon, und wählen Sie anschließend den Menüpunkt "Info" im Menü "File". Nachdem Sie den Menüpunkt angeklickt haben, erzeugt GEOS einen sogenannten Info-Bildschirm, der zahlreiche Informationen zu dem betreffenden Programm enthält. Für uns im Augenblick am wichtigsten ist die Zeile "Write Protect". Vor dem Wort "Write" befindet sich ein kleines, schwarzes Quadrat, das Sie nun bitte anklicken. Der Inhalt des Quadrats wird nun weiß, das Programm ist zum Löschen freigegeben.

Um das Programm wieder gegen Löschen zu schützen, klicken Sie einfach (aber bitte nicht jetzt!) das Quadrat noch einmal an; sein Inhalt wird dann wieder schwarz.

Um den Info-Bildschirm zu verlassen, müssen Sie das Schließsymbol in der rechten oberen Ecke (neben dem Namen des Programms) des Info-Schirms anklicken. Der alte Bildschirminhalt wird nun wiederhergestellt und gleichzeitig die durchgeführten Änderungen auf der Diskette abgelegt.

Mit den beiden anderen Programmen (GEOS KERNAL und GEOS BOOT) verfahren Sie analog; auch sie sind gegen versehentliches Löschen geschützt.

Alles weitere ist ganz einfach: Bitte klicken Sie das Icon des Programms an. Das Icon wird jetzt invertiert dargestellt. Nun klicken Sie noch einmal. Jetzt haben Sie "unter" dem Mauszeiger ein invertiertes Abbild des Programm-Icons, das Sie mit dem Joystick oder der Maus bewegen können. Fahren Sie mit dem Icon nun in die rechte untere Bildschirmecke, und bringen Sie es über den "Papierkorb". Anschließend drücken Sie nochmals den Joystick-Knopf. Das Programm wird daraufhin gelöscht, Sie haben es sozusagen in den Papierkorb geworfen. Genauso wichtig wie das Löschen ist das Kopieren von Programmen. Auch dies ist bei GEOS auf sehr elegante Weise gelöst.

Wie Sie vielleicht schon wissen, befinden sich auf der GEOS-Diskette zwei komplette Anwendungsprogramme: GEOPAINT

und GEOWRITE. GEOPAINT ist ein Mal- und Zeichenprogramm, GEOWRITE eine Textverarbeitung. Damit Sie mit diesen überhaupt sinnvoll arbeiten können, benötigen Sie eine oder am besten zwei weitere Disketten. Eine für GEOPAINT und eine für GEOWRITE.

Zunächst müssen wir die beiden Disketten formatieren. Erinnern Sie sich noch an die komplizierte Befehlszeile, die weiter oben zum Formatieren erforderlich war? Mit GEOS geht das viel einfacher: Bitte klicken Sie im Menü "Disk" den Menüpunkt "Format" an.

GEOS fordert Sie nun in einem Fenster, übrigens eine sogenannte "Dialogbox", dazu auf, die zu formatierende Diskette ins Laufwerk einzulegen und außerdem einen Namen für die neue Diskette einzugeben. Nachdem Sie dies getan und die <Return>-Taste gedrückt haben, wird die Diskette formatiert. Das dauert wie üblich etwa 90 Sekunden. Danach fordert GEOS Sie auf, wieder die alte Diskette einzulegen.

Mit der zweiten, leeren Diskette verfahren Sie analog. Als Namen für die beiden Disketten empfehlen sich "GEOPAINT" und "GEOWRITE". Sie können natürlich auch jeden anderen Namen verwenden: Hauptsache er ist nicht länger als 16 Zeichen. Jetzt kann es ans eigentliche Kopieren gehen. Zuerst kopieren wir GEOPAINT:

- Klicken Sie das Icon von GEOPAINT an. Es wird nun invertiert dargestellt.
- Klicken Sie das Icon noch einmal an. Unter dem Mauszeiger befindet sich jetzt ein Abbild des Icons, das Sie zusammen mit dem Zeiger bewegen können.
- Fahren Sie mit dem Icon zum unteren Rand des Bildschirms, der sogenannten "Border", und klicken Sie dort ein drittes Mal.

Das Icon wird nun aus dem hellen Bereich entfernt und dauerhaft auf der Border abgelegt. Das war der erste Schritt. Von der Border aus können Sie das Programm jetzt auf jede andere Dis-

kette kopieren. Dazu müssen Sie die betreffende Diskette (in diesem Fall die GEOPAINT-Diskette) nur ins Laufwerk einlegen und wie folgt verfahren:

- Öffnen Sie die Diskette, indem Sie im Menü "Disk" den Menüpunkt "Open" anklicken.
- Klicken Sie das GEOPAINT-Icon auf der Border an. Nach kurzer Pause klicken Sie ein zweites Mal. Das Icon kann jetzt wieder frei bewegt werden.
- Fahren Sie mit dem Icon nach oben in das hell unterlegte Fenster, und klicken Sie ein drittes Mal.

Das war es in Grunde genommen schon! GEOS fordert Sie nun in einer Dialogbox dazu auf, die Arbeitsdiskette ins Laufwerk einzulegen. Sobald Sie der Aufforderung nachgekommen sind, klicken Sie das OK-Icon in der Dialogbox an. Anschließend müssen Sie wieder die GEOPAINT-Diskette einlegen. Bei längeren Programmen, wie GEOPAINT, sind mehrere Diskettenwechsel erforderlich, bis das Programm komplett kopiert ist. Halten Sie sich einfach an die Anweisungen, die Ihnen GEOS gibt, und Sie können nichts falsch machen.

Nachdem das Programm kopiert ist, ist das Icon von der Border verschwunden und befindet sich im hell unterlegten Fenster der GEOPAINT-Diskette.

Kopieren Sie nun GEOWRITE auf die andere Diskette. Dazu müssen Sie zuerst wieder die Arbeitsdiskette öffnen. Vielleicht überrascht es Sie, daß das GEOPAINT-Icon jetzt wieder auf der Border auftaucht. Es steht dort für weitere Kopieraktionen zur Verfügung. Da wir es im Moment aber nicht mehr brauchen, können Sie es unbesorgt in das Fenster zurückverfrachten. Wie das geht, wissen Sie ja in der Zwischenzeit (Icon anklicken, kurz warten, nochmal klicken, Icon verschieben und wieder klicken).

Nachdem Sie GEOWRITE ebenfalls kopiert haben, sind die Arbeitsdisketten fertig. Wie Sie die Programme starten können, wissen Sie ja inzwischen (einfach ihre Icons doppelklicken). In Kapitel 2 werde ich auf GEOWRITE und GEOPAINT noch genauer eingehen. Sehen Sie sich die Programme ruhig aber schon

vorher einmal an, und versuchen Sie, die einzelnen Funktionen herauszufinden. GEOS lädt ja durch seine intuitive Bedienugsstruktur dazu ein, auf "Entdeckungstour" zu gehen, ohne groß Handbücher zu wälzen.

Neben kompletten Anwendungsprogrammen, wie die beiden eben vorgestellten, verfügt GEOS noch über eine Reihe sogenannter "Accessories", das sind kleinere Hilfsprogramme. Das Besondere an diesen Accessories: Sie lassen sich jederzeit von einem anderen Programm aus (DESKTOP, GEOPAINT oder GEOWRITE) aufrufen!

So kann man beispielsweise auf einem "Notizblock" (NOTE PAD) wichtige Termine notieren oder einen "Wecker" (ALARM CLOCK) stellen, um sich an eben diese Termine vom Rechner erinnern zu lassen.

Aufgerufen werden die Accessories über das ganz links stehende Menü der Menüleiste. Dieses Menü trägt in allen Applikationen den Namen "Geos". Klicken Sie einmal das Menü "Geos" im Desktop an.

Im unteren Bereich der Menüpunkte sehen Sie die Namen der verfügbaren Hilfsprogramme: ALARM CLOCK, CALCULATOR, NOTE PAD, PREFERENCE-MGR usw... Starten Sie versuchshalber einmal die ALARM CLOCK, indem Sie den entsprechenden Menüpunkt anklicken. Nach kurzer Zeit haben Sie eine kleine Digitaluhr mit Stunden-, Minuten- und Sekundenanzeige auf dem Bildschirm. Da es sich um eine 12-Stunden-Uhr handelt, steht hinter den Ziffern noch AM für vormittags (0 bis 12 Uhr) oder PM für nachmittags (12 bis 24 Uhr).

Im Augenblick geht die Uhr natürlich falsch. Wir müssen Sie erst noch stellen. Den Cursor, der im Moment auf dem "P" steht, können Sie mit Hilfe der Leertaste über die Ziffern bewegen, ohne etwas zu verändern. Die genaue Uhrzeit läßt sich nun mit den Zifferntasten der Tastatur einstellen. Unsinnige Angaben, wie etwa "34" für die Stunde, werden von GEOS erst gar nicht angenommen.

Wenn Sie das mittlere der drei Symbole unterhalb der Ziffernanzeige anklicken, wird die eingestellte Uhrzeit von GEOS übernommen. Durch Anklicken des rechten Symbols können Sie die Uhr verlassen.

Vielleicht möchten Sie ja aber auch eine Alarmzeit einstellen, zu der Sie GEOS durch einen Signalton "wecken" soll. In diesem Fall klicken Sie das linke Symbol an. Dessen Darstellung wechselt nun von der Uhr zum Klingelsymbol. Jetzt können Sie - wieder über die Ziffernanzeige - die Alarmzeit einstellen. Anschließend klicken Sie wieder das SET-Symbol in der Mitte und dann das rechte Schließsymbol, um die Uhr zu verlassen.

Verbesserungen in GEOS 2.0

Wie schon eingangs erwähnt, wird GEOS 2.0 auf insgesamt vier Disketten geliefert. Die Version 1.2, mit der Sie gerade arbeiten, kommt mit nur einer Diskette aus. Da muß also einiges hinzugekommen sein!

Die meisten Erweiterungen und Verbesserungen stehen in Zusammenhang mit GEOWRITE, der Textverarbeitung von GEOS. Nicht nur das Programm selbst wurde überarbeitet, es sind auch einige interessante Hilfsprogramme hinzugekommen. Beispielsweise GEOMERGE, mit dem sich sogenannte "Serienbriefe" erstellen lassen. GEOSPELL ist ein Rechtschreibprüfer, mit dem man seine Texte auf Tippfehler überprüfen lassen kann. Mit GEOLASER steht sogar ein Programm zur Verfügung, daß in der Lage ist, Texte auf einem Laserdrucker auszugeben.

Auch das DESKTOP, dessen Funktionen ich Ihnen ja gerade ausschnittsweise vorgestellt habe, wurde gründlich überarbeitet. Es unterstützt nun die Arbeit mit zwei Laufwerken und erlaubt auch den Zugriff auf die von Commodore angebotene "RAM-Erweiterung" für den Commodore 64. Die RAM-Erweiterung kann von GEOS wie ein Diskettenlaufwerk verwaltet werden, d.h., man kann auf ihr Programme oder Daten abspeichern und wieder laden. Im Gegensatz zu einer "normalen" Floppy, läuft das Ganze aber blitzschnell ab, da sich ja alles im internen Speicher des Rechners abspielt. Die RAM-Erweiterung ist aller-

dings recht teuer und dürfte daher nur für Anwender geeignet sein, die sehr intensiv mit GEOS arbeiten möchten. Ohne GEOS läßt sich mit der RAM-Erweiterung nicht allzu viel anfangen, da die meisten anderen Anwendungsprogramme keine Unterstützung dafür bieten.

Die anderen Änderungen am DESKTOP liegen mehr im Detail. So kann man nun mehrere Dateien gleichzeitig bearbeiten, etwa, um sie zu löschen oder zu kopieren. Ganz wichtig: Versehentlich gelöschte Dateien lassen sich unter gewissen Umständen wiederherstellen. Interessant ist auch die rechts oben angebrachte Uhr mit Datumsanzeige. Wer des Englischen nicht so mächtig ist, kann sich besonders freuen: GEOS 2.0 ist komplett eingedeutscht, wenn auch die deutschen Texte mitunter etwas "abgehakt" aussehen. Im Deutschen läßt sich vieles eben nicht so kurz und prägnant ausdrücken wie im Englischen.

Im nächsten Kapitel werden wir uns zunächst einmal anschauen, was man mit Hilfe kommerzieller Software mit dem Commodore 64 alles machen kann.

1.8 Anwendungen mit GEOS 2.0

Wenden wir uns nun einmal den Zusatzdisketten zu, die nicht im normalen Lieferumfang von GEOS enthalten sind, sondern zusätzlich gekauft werden können.

Dabei handelt es sich zunächst um eine Diskette, die zwei Anwenderprogramme enthält: zum einen eine Art Adressenverwaltung mit dem Namen GEODEX und zum anderen das Programm GEOMERGE, mit dem man Serienbriefe erstellen kann.

1.8.1 GEODEX

Schauen wir uns zunächst einmal an, wie GEODEX aufgebaut ist und was es bietet. Bei GEODEX handelt es sich um ein sogenanntes Accessory, d.h. ein zusätzliches Hilfsprogramm, das man

jederzeit laden kann, wenn man nicht schon ein anderes Accessory geladen hat. Man kann es also z.B. aus GEOWRITE heraus laden. Schauen wir uns nun im folgenden einmal genau an, wie man an GEODEX herankommt, vor allem beim ersten Mal, denn da sind - wie überhaupt bei den neuen GEOS-Schöpfungen - besondere Spielregeln und Gesetzmäßigkeiten zu beachten.

Zugang zu GEODEX

Auf jeden Fall müssen Sie - wie bei jedem Accessory - zunächst einmal GEOS selbst laden, bevor Sie auf GEODEX zugreifen können. Dabei müssen Sie unbedingt die Version von GEOS nehmen, mit der Sie auch in der nächsten Zeit arbeiten wollen. Denn aufgrund des neuen Kopierschutzverfahrens von GEOS können Sie alle Zusatzprogramme, die wir Ihnen hier vorstellen (GEODEX, GEOMERGE und GEOWRITE 2.0), nur mit der GEOS-Version verwenden, die Sie auch beim ersten Ausprobieren dieser Programme verwendet haben. Denn so, wie Sie GEOS V1.3 vor dem ersten Gebrauch "installieren" müssen, müssen Sie das auch mit den Zusatzprogrammen tun. Führen wir das jetzt einmal gemeinsam für GEODEX durch:

Installierung von GEODEX

Wir gehen davon aus, daß Sie GEOS geladen haben und sich im DESKTOP befinden. Nehmen Sie nun die Original-GEODEX-Diskette, entfernen Sie einen hoffentlich darauf befindlichen Schreibschutzkleber, legen Sie die Diskette ein, "öffnen" Sie sie, etwa durch Anklicken des Untermenüs "Öffnen" (OPEN) unter "Diskette" (DISK), und laden Sie anschließend GEODEX (am einfachsten durch Doppelklicken). Nach kurzer Zeit erscheint ein Fenster mit der Inschrift "GEODEX wurde installiert", und Sie können ab jetzt GEODEX ganz normal in Verbindung mit der GEOS-Version, die Sie bei dem eben beschriebenen Vorgang benutzt haben, laden, kopieren und damit arbeiten. Klicken Sie "OK" an, und nach kurzer Zeit sehen Sie wieder das DESKTOP. Bitte bringen Sie auch hier sofort wieder den Schreibschutzkleber auf, damit es auf Ihrer Original-GEODEX-Diskette keine Datenverluste gibt.

Arbeitsdiskette

Für die weiteren Experimente und Untersuchungen an GEODEX sollten Sie natürlich möglichst bald ein oder zwei Kopien der installierten GEODEX-Version anfertigen und ab sofort mit diesen Arbeitsdisketten arbeiten. Wie Sie dieses Kopien anfertigen können, ist ja weiter vorne in diesem Buch erklärt worden.

Erster Eindruck von GEODEX

Was sich nach dem Doppelklicken des GEODEX-Symbols nach kurzer Zeit präsentiert, ist ein überaus eindrucksvoller Bildschirm, der einen recht gut getroffenen Karteikasten darstellt. Im Zusammenhang damit sind alle wichtigen Zugangs- und Verwaltungsmöglichkeiten vorhanden. Schauen wir uns daher diesen Karteikasten und sein Umfeld genauer an.

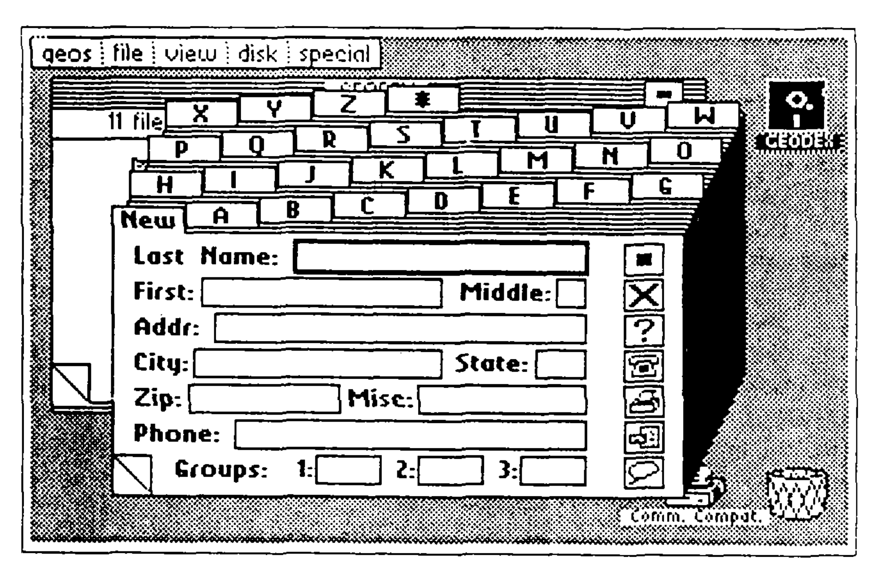


Bild 1.8.1.1: Der GEODEX-Karteikasten

Die Stirnfläche des GEODEX-Karteikastens enthält die Karteikarten-Maske, bzw. man kann dort die jeweils aktuelle Datei sehen. Über dieser Maske sieht man insgesamt 28 Registerkarten mit jeweils einem kleinen Reiter (so nennt man das wohl in der

Bürosprache), mit deren Hilfe man auf die alphabetisch geordneten Karteikarten zugreifen kann. Die Register und die Reiter sind in insgesamt vier Reihen angeordnet, wobei die unteren drei jeweils acht enthalten und in der obersten Reihe noch vier vorhanden sind.

Wichtig ist das Zeichen, das Sie in der untersten Reihe ganz links sehen (über dem Wort "last" bei "last name"). Hier erkennt man allein schon an der Anordnung, daß dieses Register zur Vorderfront-Maske gehört. Wenn Sie mit der Maus ein anderes Register anklicken, rutscht dieses in die Vorderfront, und die anderen ordnen sich entsprechend an, so daß insgesamt die richtige Reihenfolge erhalten bleibt.

Da das Alphabet nur 26 "ordentliche" Buchstaben enthält und GEODEX auf 28 Register kommt, müssen zwei zusätzliche eine besondere Bedeutung haben.

Zum einen haben wir da ein Register, auf dessen zugehörigem Reiter "NEW" steht. Dieses Register klicken Sie an, wenn Sie eine beliebige Adresse neu eingeben wollen. Dementsprechend benutzt man es auch beim Beginn der Arbeit mit GEODEX, wenn man die ersten Eintragungen vornimmt.

Das zweite aus dem Rahmen fallende Register enthält auf dem Reiter eine Art "Sternchen". Unter bzw. hinter ihm verbergen sich alle Einträge, die nicht mit einem Buchstaben, sondern z.B. mit einer Zahl oder einem anderen Zeichen beginnen. Auch die müssen ja schließlich irgendwo untergebracht werden. Verlassen wir nun zunächst einmal die "Reiterei" und wenden uns der Eingabemaske zu.

Aufbau der Maske auf der Vorderseite

Ganz oben links finden Sie ein Feld "LAST NAME", in das Sie einen Familiennamen mit maximal 13 Buchstaben eingeben können. Wenn Sie das getan haben, kommen Sie mit der <Return>-Taste zum nächsten Feld (FIRST), wo Sie bis zu 11 Buchstaben lange Vornamen eingeben können. Sollten Sie jetzt feststellen, daß im ersten Feld etwas falsch ist, können Sie mit <Cursor

links> grundsätzlich immer in das vorangehende Feld zurückkehren. Dementsprechend kommen Sie jetzt auch mit <Cursor rechts> in das nächste Feld, das den Namen "MIDDLE" trägt. Hier können Sie, falls Sie jemanden kennen, der, wie in Amerika z.T. üblich, zwischen Vor- und Familiennamen noch einen (abgekürzten) gewissermaßen "mittleren" Namen trägt, einen Buchstaben eintragen.

Falls Ihnen dies im Augenblick etwas seltsam vorkommt, so denken Sie etwa an Beispiele wie John F. Kennedy oder Richard M. Nixon.

Das nächste Feld, das Sie dann mit <Cursor rechts> oder mit Hilfe der <Return>-Taste erreichen, ist "ADDR", wo Sie zum Beispiel Straße und Hausnummer eingeben können. Dafür haben Sie einen Raum von 24 Zeichen zur Verfügung. Den Ort können Sie in das Feld "CITY" eintragen (maximal 15 Zeichen). Für deutsche Verhältnisse unnötig ist die Eintragung des Kennzeichens eines Bundesstaates in das Feld "STATE" (zwei Zeichen). Das folgende Feld "ZIP" mit seinen 10 Zeichenmöglichkeiten sollten Sie - ähnlich wie die Amerikaner - für die Eingabe der Postleitzahl benutzen. Für freie Einträge haben Sie das Feld "MISC" zur Verfügung, das auch bei den amerikanischen Benutzern dafür gedacht ist, Bemerkungen unterschiedlichster Art (maximale Länge 11 Zeichen) einzutragen (etwa "nett", oder "hat C64", was hoffentlich zusammengehört).

Was dann zu einer vollständigen Adresse noch gehört, nämlich die Telefonnummer, können Sie in das nächste Feld "PHONE" eintragen. Der dafür vorgesehene Platz von höchstens 17 Ziffern dürfte wohl für unsere Verhältnisse ausreichen (hoffentlich ist die Telefonnummer Ihres besten Freundes nicht so lang!). Ganz am Schluß haben Sie noch drei Felder mit der Bezeichnung "group", in die Sie jeweils drei Zeichen eintragen können. Sie dienen dem Sortieren und werden in diesem Kapitel später erklärt.

Hier sei aber schon darauf hingewiesen, daß es sich empfiehlt, eines der Felder, etwa das erste, für die Unterscheidung nach "männlich" und "weiblich" zu verwenden. Tragen Sie also ent-

sprechend bei jeder Adresse ein "m" oder "w" ein. Sie werden das später vor allem für Serienbriefe brauchen, wo sich etwa die Anrede bei männlichen und weiblichen Adressaten unterscheidet (vgl. dazu das Kapitel über GEOMERGE).

Falls Sie jetzt schon ein paar Namen eingeben wollen (indem Sie mit der Maus auf "NEW" gehen, falls dies nicht schon eingestellt ist), sollten wir an dieser Stelle auch bereits die Befehle auflisten, mit denen Sie sich innerhalb eines Datensatzes oder in der Kartei insgesamt bewegen können.

Dabei wiederholen wir noch einmal zum besseren Gesamtüberblick die Hinweise, die wir oben schon gegeben haben. Von einem Eingabefeld zum nächsten kommen Sie durch Drücken der <Return>-Taste. Das gleiche erreichen Sie auch durch Drücken der <Cursor rechts>-Taste. Die Gegenrichtung steht Ihnen offen, wenn Sie dementsprechend die <Cursor links>-Taste drücken.

Gespeichert wird die ganze neue Eintragung, indem Sie am Schluß oder auch aus jedem anderen Eingabefeld heraus klicken. Wollen Sie in Ihrer Kartei zurückblättern, also den vorherigen Eintrag sehen, können Sie das mit Hilfe der <Cursor hoch>-Taste erreichen. Ebenso können sie weiterblättern, indem Sie die <Cursor tief>-Taste drücken.

Dies funktioniert natürlich nur, solange Einträge vorhanden sind. Vor- und Zurückblättern können Sie auch durch das Anklicken des Eselsohrs in der Eingabemaske erreichen. Dies funktioniert so, wie Sie es auch sonst von GEOS her gewöhnt sind.

Wollen Sie die Einträge zu einem bestimmten Buchstaben sehen, so erreichen Sie das durch Anklicken des entsprechenden Reiters mit der Maus.

Gezeigt wird Ihnen dann der alphabetisch erste Eintrag unter diesem Buchstaben. Sie kommen dann zum nächsten Eintrag unter diesem Buchstaben, aber auch darüber hinaus, indem Sie wieder mit Cursor oder Eselsohr blättern. Wie oben schon erwähnt, erscheint dabei der jeweils aktuelle Buchstabe immer oben links in der Eingabemaske, und die anderen ordnen sich

entsprechend an. Kommen wir nun zu den Befehlsmöglichkeiten auf der rechten Seite der Vorderfront des GEODEX-Karteikastens, die von der Anordnung her etwas an GEOPAINT erinnern.

Die Menüleiste von GEODEX

Im folgenden sollen Ihnen die einzelnen Menüfelder in der Reihenfolge von oben nach unten vorgestellt werden. Zur leichteren Orientierung haben wir die Zahlen jeweils dazugedruckt.

Menüfeld 1: Schließsymbol

Wenn Sie dieses Feld anklicken, verlassen Sie GEODEX.

Menüfeld 2: Liegendes Kreuz

Hier können Sie den gerade aktuellen Eintrag löschen. Wenn Sie hier klicken, erscheint ein Fenster mit der Frage, ob Sie das wirklich wollen (Delete the current record?), und Sie haben die Möglichkeit, das mit "Yes" oder "No" zu beantworten.

Menüfeld 3: Das Fragezeichen

Dieses Feld dient allgemein zum Durchsuchen Ihres GEODEX-Karteikastens nach bestimmten Einträgen.

Wenn Sie hier klicken, erscheint ein Fenster mit der Inschrift "Enter last name to find:". An dieser alleinigen Aufforderung, den Familiennamen (= last name) einzugeben, nach dem Sie suchen wollen, können Sie schon erkennen, daß GEODEX nur in der Lage ist, Familiennamen nach dem Alphabet zu suchen. Nicht möglich sind zum Beispiel die Suche nach Telefonnummern oder ähnliche Sortiervorgänge (etwa das Heraussuchen aller Telefonnummern aus einem bestimmten Bereich).

Haben Sie den gesuchten Familiennamen eingegeben, müssen Sie die <Return>-Taste drücken, und schon geht die Sucherei los. Findet GEODEX zu dem Namen, den Sie eingegeben haben, keinen Eintrag, dann erscheint ein Fenster mit der Meldung:

"There ist no one with that last name in the GEODEX file." Und anschließend können Sie eine neue Suche ausprobieren.

(Übrigens: Falls Sie schon mit den "group"-Feldern gearbeitet haben, kann die erfolglose Suche natürlich auch daran liegen, daß GEODEX (siehe das unterste Feld "Lassoschlinge") nur noch nach Einträgen sucht, die entsprechende Einträge in den "group"-Feldern haben. Sie müssen dann dort erst wieder in den Modus gehen, bei dem alle Daten berücksichtigt werden.)

Probieren Sie das Suchen nach Einträgen jetzt ruhig einmal bei etwa drei oder vier Einträgen selbst aus.

Vielleicht haben Sie schon selbst dabei an die Verwendung des Jokers * gedacht, den Sie ja von der tagtäglichen Nutzung Ihres C64 her kennen. Und tatsächlich, er funktioniert auch hier. Sie können damit den Rest des gesuchten Namens weglassen.

Suchen Sie also etwa nach "Mei*", so finden Sie unter Umständen (je nach Ihren Einträgen natürlich) Meier, Meiers, Meisterkamp usw... Wenn Sie auf den Joker allerdings verzichten, sucht GEODEX genau nach der Zeichenkombination, die Sie eingegeben haben.

Aber es gibt noch eine weitere, gerade im Falle unseres Herrn Meier nicht uninteressante Möglichkeit. Stellen wir uns vor, Sie wissen nicht mehr, ob Ihr "Meier" mit "i" oder mit "y" geschrieben wird. Hier kann GEODEX Ihnen helfen, wenn Sie anstelle des unbekannten Buchstabens ein Fragezeichen einfügen.

Suchen Sie also nach "Me?er", so findet Ihr Programm entweder Meier oder Meyer - oder beides. "Meier" ist insofern ein gutes Beispiel, weil es ja im Umfeld dieses Namens Leute gibt, die sogar so weit gehen, sich mit "ay" zu schreiben.

Wollen Sie auch diesen Leuten eine Chance geben, zumindest gemeinsam erfolgreich gesucht zu werden, so geben Sie doch einfach zwei Fragezeichen an der Stelle ein, wo Sie sich alle Möglichkeiten offenhalten wollen. Am Ende steht also im Suchfeld "M??er".

Und wenn Sie jetzt einen "Meier", einen "Meyer", einen "Mayer" oder gar einen "Maier" in Ihrer Kartei haben, werden sie gefunden, wetten?!

Menüfeld 4: Telefon

Um dieses Feld erfolgreich benutzen zu können, benötigen Sie einen Akustikkoppler, der amerikanischem Standard entspricht. Sollten Sie das nicht glauben, klicken Sie dieses Feld doch einfach einmal an.

Schnell erscheint ein Fenster, auf dem Ihnen das Programm zu seinem Kummer mitteilt: "Cannot access modem to dial" (zu Deutsch etwa: "Ich komm' nicht ins Telefonnetz rein!").

Menüfeld 5: Drucker

GEODEX bietet Ihnen eine ganze Palette von Druckmöglichkeiten. Wenn Sie dieses Feld anklicken, erscheint in einem Fenster ein Auswahlmenü mit der Inschrift "Select desired output", die Sie einfach auffordert, eine Wahl zu treffen.

Als erste Möglichkeit können Sie "Phone list" anklicken. Dann werden linksbündig Vorname und Nachname und rechtsbündig die Telefonnummer gedruckt, der Rest der Zeile wird durch Pünktchen aufgefüllt.

Wählen Sie "Adress labels (1 inch)", so werden jeweils Vorname und Nachname in die erste Zeile, die Adresse, d.h. bei uns die Straße, in die nächste Zeile und Ort, Staat und ZIP-Postleitzahl in die dritte Zeile gedruckt. Wählen Sie "Adress labels (1 1/2 inch)", so ergibt sich nur ein anderer Zeilenabstand zwischen den Angaben. Klicken Sie "list all field data" an, so werden alle Daten der Karteikarte gedruckt.

Und zwar geschieht das in folgender Reihenfolge:

1. Zeile: Vorname, Middle, Familienname und (rechtsbündig) Telefonnumer
2. Zeile: Adresse = Straße

3. Zeile: Ort, Staat, ZIP-Postleitzahl
4. Zeile: der Eintrag unter "MISC". sowie die Einträge in den "group"-Feldern

Die von Ihnen angewählte Ausgabeart bekommt, wie Sie gesehen haben, jeweils ein schwarzes Kästchen.

Sehr komfortabel ist, daß es sich um einen Textmodusdruck, nicht um Grafikdruck handelt, das heißt, der Druck ist sehr schnell.

Sie können daher GEODEX nur mit den neuen Druckertreibern verwenden, die sich auf der Rückseite der GEODEX-Diskette befinden, da nur sie im Textmodus drucken können. Der Druck beginnt, sobald Sie "OK" angeklickt haben.

Menüfeld 6: Bild einer Karteikarte und eines Blattes Papier

Wenn Sie dieses Feld wählen, erscheint ein Fenster mit der Aufforderung "Enter merge file name", d.h., Sie sollen also den Namen einer Datei eingeben, die Sie jetzt herstellen und die es Ihnen ermöglicht, Serienbriefe zu schreiben.

Wir werden auf diese Möglichkeit näher im Zusammenhang mit GEOMERGE eingehen.

Menüfeld 7: Lasso

Hier werden Sie aufgefordert, eine "group" anzugeben: "Enter the group to view".

GEODEX zeigt Ihnen jetzt nur noch die Adresseneinträge in einem der drei "group"-Felder, die sich ganz unten auf jeder Karteikarte befinden. Auf diese Art und Weise können Sie also Ihre gesamten Karteikarten in Unterkarteien ordnen.

Nehmen wir einmal an, Sie möchten gerne bei Ihren Adressen zwischen denen privater und denen beruflicher Natur unterscheiden (etwa weil Sie Ihren Chef nicht unbedingt per Serienbrief zu Ihrer Samstagabendfete einladen wollen). Dann sollten

Sie allen denjenigen, die Sie zu Ihren privaten Bekannten oder Freunden zählen, etwa die Abkürzung "pri" für "privat" in eines der Gruppenfelder schreiben.

Noch einmal: Es ist dabei gleichgültig, in welchem der drei Gruppenfelder die Eintragung auftaucht. Das Feld, nach dem gesucht wird, wird invertiert, und schon sehen Sie nur noch die Einträge Ihrer Kartei, die die "group"-Bedingung erfüllen. Das Blättern in dieser eingeschränkten Kartei geschieht auf die oben erklärte Weise (Cursor-Steuerung oder Eselsohr-Klick).

Wollen Sie hinterher den Gruppen-Sortiermodus verlassen, gehen Sie wieder auf das Lassofeld und klicken "View all" an. Und schon sind Sie wieder im Normal-Anzeigemodus.

Mit "Cancel" können Sie sich übrigens - wie immer - aus einem Fenster entfernen, wenn Sie es sich nur anschauen, aber in diesem Falle keine Gruppe angeben wollten.

Übersicht über die Möglichkeiten bei der Tastatureingabe

Im folgenden möchten wir Ihnen noch einmal einen Gesamtüberblick über die Eingabemöglichkeiten bei GEODEX geben. Wie Sie schon gesehen haben, haben Sie bei der Eingabe von Daten in die Felder alle Zeichentasten zur Verfügung, außerdem die <Del>-Taste, um Zeichen zu löschen.

Was nun die Befehlseingaben angeht, so finden Sie hier eine Tabelle der Tastenbelegung bei GEODEX, wobei es uns vor allem auf Kurzbefehle ankommt. Aber wir präsentieren Ihnen auch noch einmal einige andere wichtige Tasten (in alphabetischer Reihenfolge).

Cursor

Cursor hoch

Wechsel zur vorhergehenden Karteikarte

Cursor tief

Wechsel zur nachfolgenden Karteikarte

Cursor rechts

Wechsel zum nächsten Eingabefeld innerhalb der Maske

Cursor links

Wechsel zum vorausgehenden Eingabefeld innerhalb der Maske

Maus

Direktwahl des alphabetischen Bereichs durch Anklicken des entsprechenden Reiters, außerdem Wahl der Menüfunktionen sowie Blättern mit dem Eselsohr an der Vorderseite des Karteikastens.

<Return>-Taste

Wechsel zum nächsten Eingabefeld innerhalb der Maske, außerdem Ausführung der Funktionen innerhalb der Fenster des Menübereichs, wo Sie keine Anklick-Möglichkeit finden.

Und nun die Kurzbefehle (in alphabetischer Reihenfolge):

<Commodore>-Taste und <D>

Telefon wählen (bei uns nicht zu gebrauchen)

<Commodore>-Taste und <G>

Gruppe auswählen (groups)

<Commodore>-Taste und <M>

Merge-File erzeugen (vgl. dazu die Ausführungen oben)

<Commodore>-Taste und <P>

Drucken

<Commodore>-Taste und <Q>

GEODEX verlassen

<Commodore>-Taste und <X>

Aktuellen Eintrag ("record") löschen

<Commodore>-Taste und <S>

Suchen nach einem Dateieintrag

<Commodore>-Taste und <Y>

Style wählen zwischen "plain" und "bold" (Normalschrift oder Fett, wird auf Bildschirm nicht angezeigt)

<Commodore>-Taste und <Shift> und Buchstabe von A-Z

Bewirkt das Suchen nach Einträgen zu dem entsprechenden Anfangsbuchstaben (entspricht dem Anklicken eines Reiters mit der Maus)

Liste der Fehlermeldungen

Im folgenden möchten wir Ihnen eine weitere tabellarische Hilfe an die Hand geben, indem wir alle englischen Fehlermeldungen zusammenstellen und gleichzeitig übersetzen bzw. erklären:

There is insufficient memory to save more than 24 names that start with that letter

Es gibt nicht genügend Speicher, um mehr als 24 Namen mit demselben Buchstaben zu speichern

Lösung: Kopieren Sie GEODEX und GEOMERGE auf eine andere Arbeitsdiskette. Sie erhalten dadurch einen weiteren (leeren) Karteikasten

There is insufficient room to load the printer driver

Es gibt nicht genügend Speicherplatz, um den Druckertreiber zu laden

Lösung: Löschen Sie nicht benötigte Files auf der Diskette.

There is insufficient room to save the merge file

Es gibt nicht genügend Speicherplatz, um das Merge-File auf der Diskette zu erzeugen

Lösung: Löschen Sie nicht benötigte Files auf der Diskette.

There is insufficient room to save the previous entry

Es gibt nicht genügend Speicherplatz, um den vorhergehenden Eintrag zu speichern

Lösung: Löschen Sie nicht benötigte Files auf der Diskette.

There is no one with that last name in the geodex file:

Es gibt in der Datei keinen Eintrag mit diesem Familiennamen

Lösung: Erstellen Sie doch einfach eine entsprechende Karteikarte, dann kann GEODEX sie auch finden.

Im übrigen prüfen Sie, ob Sie sich nicht in einem "group"-Auswahlmodus (Menüpunkt "Lassoschlinge") befinden, der es GEODEX unmöglich macht, Ihnen einen Eintrag zu präsentieren, den Sie zwar haben, der aber Ihre gewählte "group"-Bedingung nicht erfüllt. Sollte das die Lösung für Ihr Suchproblem sein, klicken Sie unter "Lassoschlinge" wieder (???) an.

1.8.2 GEOWRITE 2.0

Im folgenden Kapitel wollen wir Ihnen nun die neueste uns bekannt gewordene Version von GEOWRITE vorstellen. Sie trägt die Entwicklungsnummer 2.0, was schon andeutet, daß hier massivere Änderungen gegenüber den verschiedenen 1.-Versionen vorgenommen worden sind.

Überblick über die Diskette

Die Diskette, auf der sich GEOWRITE 2.0 befindet, heißt "Writer's Workshop" und zeigt damit schon ihren Anspruch, einen kompletten Arbeitsplatz für jemanden zu bieten, der, sei es privat - sei es beruflich - viel schreibt.

Auf der Vorderseite der Diskette finden sich auf der ersten Seite des Inhaltsverzeichnisses neben GEOWRITE 2.0 noch das schon in diesem Buch beschriebene Serienbrief-Programm GEOMERGE, außerdem ein Programm mit Namen TEXT GRABBER, wobei es sich hier um ein Programm zur Konvertierung von Texten ins GEOWRITE-Format handelt, und schließlich GEOLASER, mit dem Sie GEOWRITE-Texte mit einem Laserdrucker ausgeben können.

Auf der zweiten Seite des Inhaltsverzeichnisses sehen Sie 5 Icons, die alle ein Commodore-Symbol tragen, verbunden mit einigen Blättern Papier. Unter diesen finden Sie die Bezeichnungen PAPERCLIP FORM, EASYSCRIPT FORM, PEEDSCRIPT FORM, ORDWRITER FORM und schließlich GENERIC FORM. Hinter diesen fünf Icons verbergen sich konkrete Anpassungen für den TEXT GRABBER.

Auf der dritten und letzten Seite des Inhaltsverzeichnisses finden sich dann der Druckertreiber LASER WRITER und fünf neue Fonts, die alle ein LW und dann einen Unterstrich haben, also LW_ROMA, LW_CAL, LW_GREEK und LW_BARROWS. Das sind die konkreten Fonts für den Laserdrucker.

Der erste Eindruck von GEOWRITE 2.0

Schauen wir uns jetzt GEOWRITE 2.0 einmal konkret an, wobei es uns nur auf die wesentlichen Unterschiede zu GEOWRITE 1.2 und GEOWRITE 1.3 ankommt. Nachdem wir GEOWRITE 2.0 geladen und das erste Fenster, wie wir es gewöhnt sind, durch die Eingabe eines Textnamens übersprungen haben, sehen wir einen Bildschirm, der grundsätzlich an die früheren Versionen erinnert, aber bei näherem Hinsehen zwei auffällige Unterschiede präsentiert.

1. Die Menüleiste oben enthält mehr Oberbegriffe:
   Hier finden sich nun Menüs mit den Namen GEOS, FILE, EDIT, OPTIONS, PAGE, FONT, STYLE.
2. Unter der Positionsleiste mit Tabulatoren findet sich noch eine weitere Zeile mit den Begriffen LEFT, CENTER, RIGHT, FULL, jeweils mit Anklick-Kästchen versehen, JUSTIFICATION mit einem vorangestellten Pfeil nach links, LINE SPACING mit einem nachgestellten Pfeil nach rechts, des weiteren die Möglichkeiten 1, 1 1/2, 2, wiederum mit einem Kästchen, das nach Anklicken durch dunkle Färbung anzeigt, daß es aktiviert worden ist. Diese neue Zeile wollen wir im folgenden "Befehlsleiste" nennen.

Erste Schreibversuche mit GEOWRITE 2.0

Die nächsten Neuerungen tauchen auf, wenn wir ein paar Zeilen auf den Bildschirm schreiben und dann den Cursor benutzen. Grundsätzlich entspricht die Cursor-Bewegung dabei erst einmal GEOWRITE 1.3.

Neu ist aber, daß man an das Ende der vorherigen Zeile gelangen kann, wenn man am Anfang einer Zeile steht und die <Cursor links>-Taste drückt. Steht man am Ende einer Zeile, kann man durch Drücken der <Cursor rechts>-Taste an den Anfang der nächsten Zeile gelangen.

Neu ist auch die Belegung der beiden anderen Cursor-Bewegungen. Wenn man mit dem Cursor an den oberen Rand geht und darüber befindet sich noch Text, dann scrollt der Text nach oben. Entsprechendes gilt für die andere Richtung. Scroll-Pfeile braucht man jetzt bei der Textverarbeitung mit GEOWRITE nicht mehr, und so gibt es verständlicherweise auch keine Scroll-Pfeile mehr, wie wir beim ersten Betrachten des GEOWRITE-2.0-Bildschirms schon feststellen konnten. Es gibt nur noch den Seitenzeiger, der einem anzeigt, wo man sich auf der ganzen Seite befindet.

Schauen wir uns nun als nächstes die Maus an. Wir können mit ihr wieder den Cursor überall hinklicken, wo Text ist. Wenn wir den Cursor auf einer Zeile hinter dem Text "anklicken", wird er

automatisch hinter das letzte Zeichen gesetzt. Geht man mit der Maus ganz zum oberen oder unteren Bildschirmrand, beginnt dort derselbe Scroll-Vorgang, wie wir ihn bereits vom Cursor her kennen - unter der Voraussetzung, daß dort noch Text zu finden ist.

Insgesamt ist damit durch die neuen Möglichkeiten des Cursors und der Maus das Arbeiten auf einer Seite wesentlich komfortabler geworden. Allerdings haben diese Verbesserungen gleichzeitig auch eine - allerdings kleine - Kehrseite.

Wenn man schon oberhalb des Bildschirms Text hat und einen Menüpunkt anklicken will, dann kann es einem plötzlich passieren, daß man nicht den gewünschten Menüpunkt erwischt, sondern vorher schon der Text anfängt zu scrollen.

Deswegen sollte man mit der Maus nicht einfach nach oben fahren und irgendwo klicken, sondern wirklich nur bis zum Anfang der Menüleiste und dann durch gezieltes Klicken das entsprechende Pulldown-Menü herunterlassen. Schauen wir uns als nächstes die Menüleiste mit ihren jeweiligen Untermenüs an.

Menü 1: GEOS

Hier hat sich nichts geändert, alle Accessories erscheinen, dazu das GEOWRITE-Info, das wie üblich die Autoren von GEOWRITE 2.0 nennt.

Menü 2: FILE

Der nächste Punkt, FILE, hat sich auch nicht geändert: Dort finden sich so bekannte Untermenüs wie CLOSE, UPDATE usw... Leider gibt es hier keine Kurzbefehle.

Menü 3: EDIT

Unter EDIT finden wir wieder CUT, COPY und PASTE, hier finden sich auch wieder Abkürzungen über Tastaturbefehle, wie wir sie schon von GEOWRITE 1.3 her kennen.

Menü 4: OPTIONS

Hier wird es nun besonders interessant, denn es tauchen die vielfältigen Möglichkeiten der Textverarbeitung auf, die man bei den früheren Versionen GEOWRITE 1.2 und GEOWRITE 1.3 immer vermißt hat. Im einzelnen sieht man zunächst folgende Untermenüs:

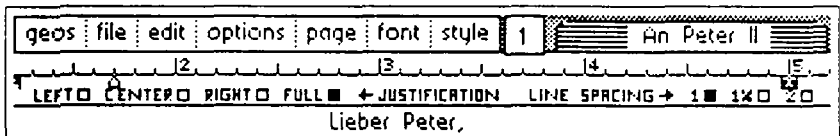
```
SEARCH
FIND NEXT
CHANGE, THEN FIND
HIDE PICTURES
OPEN HEADER
OPEN FOOTER
SELECT PAGE
```

Bis auf HIDE PICTURES sind alle über Tastaturabkürzungen zu erreichen. Um die Möglichkeiten ausprobieren zu können, brauchen wir hier natürlich einen schon etwas längeren Text.

Wenn Sie unseren Versuchen folgen wollen, sollten Sie jetzt den folgenden Brief unter dem Namen "An Peter II" erstellen:

```
Lieber Peter,
Dies ist mein erster Brief mit dem neuen GEOWRITE 2.0, das mir Herr
Meier empfohlen hat. Eben habe gelesen, daß man jetzt endlich
professionelle Textverarbeitung machen kann. Ich bin gespannt, wie
das im einzelnen so aussehen wird.
Es ist doch erstaunlich, was aus dem alten C64 so noch alles
herausgeholt worden ist. Wer hätte das vor ein, zwei Jahren gedacht,
daß es für ein 64-KByte-Gerät einmal eine Benutzeroberfläche geben
würde.
Ich bin doch froh, daß ich damals auf Herrn Meier gehört und mich für
Commodores Renner entschieden habe.
Am meisten aber bin ich darüber froh, daß es nicht bei dem einmaligen
Wurf geblieben ist, sondern daß ständig Verbesserungen und
Erweiterungen zu GEOS kommen.
So, jetzt habe ich genug Text für meine Versuche, ich verabschiede
mich daher zunächst einmal,


Bis bald
   Dein Gerd
```



Lieber Peter,

Dies ist mein erster Brief mit dem neuen GEOWRITE 2.0, dass mit Herr Meier empfohlen hat. Eben habe ich im GROSSEN GEOSBuch gelesen, dass man jetzt endlich professionelle Textverarbeitung mit GEOS machen kann. Ich bin gespannt, wie dass im einzelnen so aussehen wird.

Es ist doch erstaunlich, was aus dem alten C64 so noch alles herausgeholt worden ist. Wer haette das vor 1-2 Jahren gedacht, dass es fuer ein 64-Kbytes-Geraet einmal eine Benutzeroberflaeche geben wuerde.

Ich bin doch froh, dass ich damals auf Herrn Meier gehoert und nicht fuer Commodores Renner entschieden haben.

Am meisten aber bin ich darueber froh, dass es nicht bei dem

Bild 1.8.2.1: Brief an Peter (GEOWRITE 2.0)

Wählen wir unter OPTIONS zunächst das Untermenü SEARCH an. Nun erscheint ein Fenster mit der Überschrift "Search/Replace". Darunter finden sich zwei Zeilen, in die man zunächst das zu suchende ("Search For") und anschließend das Wort eingeben kann, das anstelle des ersten erscheinen soll ("Replace "with"). Der Cursor blinkt bereits in der ersten Zeile.

Also geben wir doch einfach zur Probe "Brief" ein. Achten Sie hierbei darauf, daß der Cursor am Anfang des Textes steht.

Wenn wir jetzt die <Return>-Taste drücken, dann sucht GEOWRITE 2.0 und markiert das gefundene Wort, indem es dieses invertiert, wie wir es auch mit der Maus tun können. In unserem Fall geschieht das in der ersten Zeile.

Als nächstes wollen wir "Brief" durch "Versuch" ersetzen. Dafür gehen wir mit dem Cursor wieder an den Anfang des Textes, wählen unter OPTIONS das Untermenü SEARCH an und klicken nun das zweite Eingabefeld an.

Hinweis: Versuchen Sie nicht, das erste Feld mit Hilfe der <Return>-Taste zu verlassen, wie Sie es von GEODEX her gewohnt sind, denn dann beginnt GEOWRITE 2.0 bereits zu suchen. Klicken Sie also mit der Maus das REPLACE-WITH-Feld an, und tippen Sie dort "Versuch" ein.

In zwei weiteren Zeilen des Eingabefensters können wir jetzt wählen zwischen WHOLE WORD und PARTIAL WORD. D.h., im ersten Falle sucht GEOWRITE 2.0 so lange, bis es genau die eingetippte Zeichenkombination als Wort vorfindet. Im zweiten Falle markiert es jedes Wort, in dem sich die Zeichenkombination findet, auch wenn das ganze Wort länger ist. Nehmen wir unser Beispiel:

Im ersten Fall würde nur "Brief" gefunden und dann gegebenenfalls ersetzt, im zweiten Falle etwa auch ein Wort wie "Briefmarke". (Auf die Wortschöpfung "Versuchsmarke" verzichten wir natürlich gerne.). In der nächsten Auswahlzeile können wir GEOWRITE 2.0 den Umfang seiner Arbeit signalisieren. Ist das linke Feld ALL PAGES angeklickt, so werden wirklich alle Seiten des Textes durchsucht, im anderen Fall "THIS PAGE ONLY" nur die gerade aktuelle Seite. Ganz unten in unserem Eigabefenster finden wir noch drei Anklickmöglichkeiten:

NEXT, ALL und CANCEL.

Wenn Sie kein ALL sehen, so liegt das nur daran, daß Sie noch nichts unter REPLACE WITH eingetragen haben.

NEXT: GEOWRITE sucht das nächste passende Wort ab Cursor-Position.

ALL: GEOWRITE sucht alle passenden Wörter ab Cursor-Position.

CANCEL: Mit diesem Feld können Sie wie immer die Eingabemaske verlassen, ohne GEOWRITE 2.0 irgendwelche Suchaufträge zu erteilen.

Wir klicken in unserem Beispiel "Brief/Versuch" die Felder WHOLE WORD, ALL PAGES und ALL an, und siehe da, es klappt: In der ersten Zeile steht jetzt "Dies ist mein erster Versuch..."

Probieren wir nun die Möglichkeiten gleich noch einmal aus, indem wir uns daran erinnern, daß Herr Meier eigentlich mit "ay" geschrieben wird. Diese Änderung können wir nun vornehmen, indem wir:

1. den Cursor an den Anfang des Textes setzen,
2. unter OPTIONS SEARCH anwählen,
3. in das erste Feld "Meier" eingeben,
4. Mit der Maus das zweite Feld anwählen und dort Mayer eintragen.
5. Außerdem wählen wir jetzt PARTIAL WORD, damit auch eine Wendung wie "Meiers Vorschlag" ersetzt wird.
6. Zusätzlich wählen wir ALL PAGES und ganz unten ALL an.

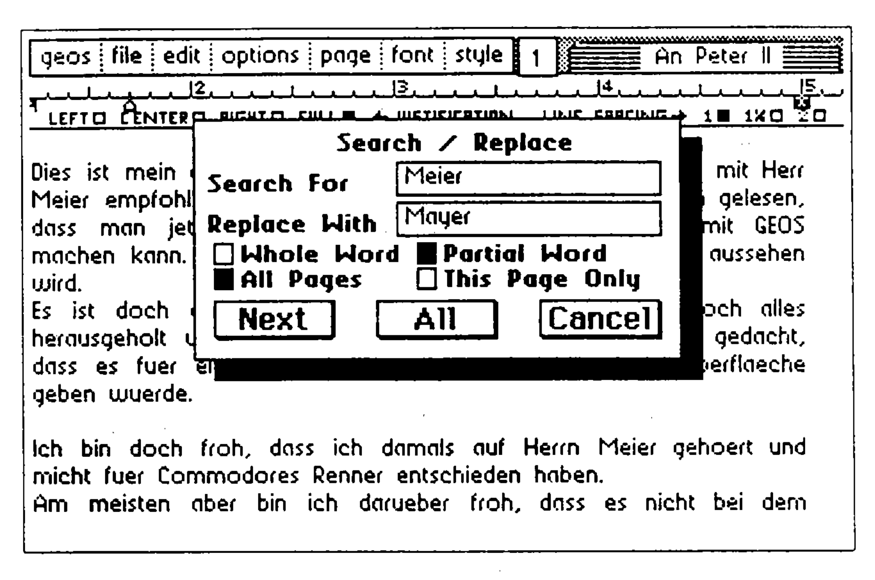


Bild 1.8.2.2: Suchen und Ersetzen mit GEOWRITE 2.0

Während jetzt Ihr Diskettenlaufwerk arbeitet, möchten wir Sie beruhigen: Daß der Vorgang so kompliziert aussieht (6 Punkte),

hängt nur damit zusammen, daß wir Ihnen die Möglichkeiten im einzelnen erklären wollten. Normalerweise sind die Auswahlmöglichkeiten schon richtig voreingestellt, und Sie haben sich rasch an den Ablauf gewöhnt.

Zum Schluß dieses Abschnitts noch ein paar kleine Tips:

1. GEOWRITE 2.0 beachtet beim Suchen Groß- und Kleinschreibung, d.h., Sie müssen dementsprechend sorgfältig bei der Eingabe des zu ersetzenden Wortes sein.
2. Wenn Sie unter "Replace" etwas eingetragen haben, sollten Sie zum Schluß NEXT oder ALL anklicken. Nur dann ersetzt GEOWRITE in Ihrem Sinne. Wenn Sie dagegen die <Return>-Taste drücken, dann sucht GEOWRITE 2.0 nur und markiert die gewünschte Stelle, ohne die Veränderung auch durchzuführen.
3. Beachten Sie außerdem beim Suchen, daß der Cursor möglichst am Anfang des Textes steht, denn gesucht wird immer ab Cursor-Position!

Kommen wir nun zu den weiteren Untermenüs von OPTION:

Untermenü: FIND NEXT

Wenn GEOWRITE 2.0 ein Wort gefunden hat und darstellt, dann kann man es mit FIND NEXT übergehen und das nächste suchen lassen.

Voraussetzung ist natürlich, daß in der SEARCH-Maske etwas eingegeben ist, sonst geschieht wie auch beim nächsten Punkt gar nichts.

Untermenü: CHANGE, THEN FIND

Hier wird im Gegensatz dazu das gefundene Wort erst ersetzt, bevor weitergesucht wird.

Somit hat man ein kontrolliertes und bestätigtes Ersetzen. Über Tastatureingaben kann dieser Vorgang abgekürzt werden.

Untermenü: HIDE PICTURES

Dieses Untermenü kennen wir schon von den früheren Versionen von GEOWRITE.

Wenn Grafiken in den Text eingeklebt sind, dann braucht GEOWRITE 2.0 die Bilder nicht anzuzeigen was unnötig Zeit kosten würde. Das Programm reserviert nur die entsprechende freie Fläche, markiert ein Rechteck an die Stelle, wo später die Grafik ausgedruckt wird.

Das spart Zeit und Diskettenzugriff. Nach dem Anklicken von HIDE PICTURES steht dort SHOW PICTURES, womit Sie die eingeklebten Grafiken wieder sichtbar machen können.

Untermenü: OPEN HEADER

Hier können Sie einen Kopfbereich definieren. Wenn Sie dieses Feld anklicken, wird Ihr Bildschirm freigemacht, unten wird ein kleiner Bereich durch einen durchgehenden, waagrechten schwarzen Strich abgetrennt, und darunter steht in großen Buchstaben "HEADER". Oberhalb der schwarzen Linie können Sie nun die Eingaben machen, die Sie auf jeder Seite als Kopfzeile bzw. Kopfzeilen vorfinden wollen. Tragen wir also in unserem Beispiel ein "An Peter II". Wir tun jetzt einfach einmal so, als ob der Brief an Peter noch sehr stark ausgebaut würde. (Das hätte er natürlich wie GEOS selbst durchaus verdient!)

Besonders interessant sind jetzt einige weitere Möglichkeiten, die GEOWRITE 2.0 bietet. So kann z.B., was ja bei einem Seitenkopf besonders sinnvoll ist, in großen Buchstaben PAGE eingegeben werden. Dann wird an dieser Stelle immer die aktuelle Seitennummer ausgedruckt. Bevor wir das nun aber abschließen und zur Probe einmal ausdrucken, sollten Sie noch die weiteren Möglichkeiten zumindest zur Kenntnis nehmen. Sie können neben PAGE nämlich noch DATE und TIME eingeben. (Bitte unbedingt auf die großen Buchstaben achten!) Auf jeder Seite werden nun das aktuelle Datum und die aktuelle Zeit ausgedruckt, falls Sie die Werte richtig zu Beginn Ihrer täglichen Arbeit mit GEOS eingestellt haben.

Insgesamt haben Sie für Eingaben im HEADER mehrere Zeilen zur Verfügung. Wenn Sie mit Ihren Eingaben fertig sind, gehen Sie bitte wieder auf OPTIONS, wo Sie jetzt - das Programm ist halt "intelligent" - einen CLOSE HEADER vorfinden. Wenn Sie diesen angeklickt haben, verschwindet der gesamte Kopfbereich, und Sie können Ihren Brief o.ä. weiterschreiben.

Wundern Sie sich bitte nicht, wenn Sie beim Scrollen an den Anfang des Textes Ihren Kopfbereich nicht sehen, es handelt sich um Informationen, die nur für den Druckvorgang gedacht sind. Nach dem Ausdrucken werden Sie es auf jeder Seite vorfinden.

Wenn Sie übrigens Ihre erste Seite, also die Titelseite, nicht mit dem HEADER bedrucken wollen, sei schon hier darauf hingewiesen, daß Sie das erreichen, wenn Sie unter PAGE den Programmpunkt TITLE PAGE anklicken. Was das Löschen des Kopfbereichs angeht, lesen Sie bitte entsprechendes unter OPEN FOOTER nach.

Untermenü: OPEN FOOTER

Wenn Sie nun pro Seite nicht nur oben immer einen abgetrennten und sich bis auf die Seitennummer wiederholenden Bereich haben wollen, sondern auch jeweils unten etwas Entsprechendes präsentieren wollen, so können Sie das ganz sinngemäß machen, indem Sie unter OPTIONS den Programmpunkt OPEN FOOTER anklicken und nach Ihren Eingaben das ganze wieder durch CLOSE FOOTER abschließen.

Übrigens können Sie auch hier wieder auf PAGE, DATE und TIME zurückgreifen. Dies ist ja auch sinnvoll ist, arbeiten doch auch viele Bücher erfolgreich mit Seitenangaben unten auf den Blättern.

Auch für den FOOTER, also Ihren "Fußbereich" gilt, daß Sie ihn erst nach dem Ausdruck sehen können.

Hinweis: Wenn Sie Ihren Fußbereich immer ganz unten auf der Seite haben wollen, müssen Sie, falls der Text

nur einen kleinen Teil des Blattes füllt, den Rest durch Leerzeilen auffüllen. GEOWRITE 2.0 setzt den Fußbereich nicht automatisch ganz nach unten.

Sollten Sie Ihren Kopf- und/oder den Fußbereich wieder entfernen wollen, geht das nur durch Löschen des entsprechenden Textes, nachdem Sie wieder OPEN FOOTER bzw. OPEN HEADER angeklickt haben. Wenn Sie dann mit CLOSE FOOTER bzw. CLOSE HEADER diese Bereiche verlassen, haben Sie beim Ausdruck keine sich wiederholenden Bereiche mehr.

Untermenü: SELECT PAGE

Mit diesem Programmpunkt können Sie die ganze Seite markieren, um sie z.B. in der Schrift zu ändern. Dies ist besonders interessant, wenn Sie unserem Ratschlag gefolgt sind und zunächst eine kleine Schrift verwenden, um möglichst viel zu sehen, bevor Sie dann auf die gewünschte Schrift umschalten.

Sehr angenehm ist in diesem Zusammenhang auch, daß Sie beim Markieren von Textpassagen mit der Maus nicht mehr auf den sichtbaren Text beschränkt sind. Im Gegensatz zu GEOWRITE 1.2/1.3 scrollt GEOWRITE 2.0 nämlich, wenn Sie beim Markieren von Text an den unteren Bildschirmrand kommen.

Menü: PAGE

Mit PREVIOUS PAGE können Sie auf die vorherige Seite umschalten, mit NEXT PAGE dementsprechend auf die folgende. Wenn Sie GOTO PAGE anklicken, erscheint ein Fenster, wo die gewünschte Zahl eingegeben werden kann.

PAGE BREAK erzeugt Ihnen eine neue Seite, die alte wird mit einem waagrechten Strich auf dem Bildschirm abgeschlossen. Die nächsten drei Untermenüs können Sie nicht durch Tastaturbefehle abkürzen:

Mit SET FIRST PAGE haben Sie die Möglichkeit, den Ausdruck Ihres Textes mit jeder beliebigen Seitennummer beginnen zu lassen. Voraussetzung ist natürlich, daß Sie eine Kopfzeile defi-

niert haben, die PAGE enthält. Wichtig ist dies, wenn Sie größere Texte in Teilen ausdrucken lassen wollen und jeder Teil natürlich nicht wieder mit Seite 1 beginnen, sondern sich mit der Seitennummer an das vorangehende Kapitel anschließen soll.

Wenn Sie SET FIRST PAGE anklicken, erscheint ein Fenster mit der Bitte um Eingabe der Seitennummer, mit der Sie beginnen wollen ("Please enter page number for first page!"). Anschließend drücken Sie die <Return>-Taste. Wenn Sie später den Text ausdrucken, beginnt er mit der eingegebenen Seitennummer.

TITLE PAGE gibt Ihnen die Möglichkeit, die erste Seite Ihres Textes ohne Kopf- und Fußbereich ausdrucken zu lassen, was aus optischen Gründen (Titelblatt) oft sehr sinnvoll ist. Wenn Sie diesen Programmpunkt anklicken, erscheint davor ein Sternchen, das Ihnen anzeigt, daß er aktiviert ist.

Beachten Sie bitte, daß GEOWRITE 2.0 diese Anweisung wirklich nur für die erste Seite akzeptiert. Das heißt, wenn Sie Ihr zweites Kapitel mit SET FIRST PAGE beispielsweise auf S. 15 starten lassen, erscheint dort auf S. 15 trotz TITLE PAGE der Kopf- bzw. Fußbereich.

NLQ SPACING sollten Sie aber erst aktivieren, bevor Sie im Druckermenü "NLQ-Druck" anwählen. Näheres können wir Ihnen zu diesem Punkt leider nicht sagen, weil dieser Programmpunkt bei unserem FX85 nicht funktioniert hat: GEOWRITE begann dann, leere Blätter zu poduzieren, bis wir den Rechner abstellten. Vielleicht haben Sie mehr Glück.

Was Sie auf jeden Fall auch beachten sollten: Für "NLQ-Druck" muß der gesamte Text mit dem FONT "COMMODORE 10" geschrieben sein.

Was auch noch zu beachten ist: Der NLQ-Druck funktioniert nur mit den neuen Druckertreibern, die Sie zusammen mit GEOWRITE 2.0 bekommen. Denn nur diese neuen Druckertreiber verfügen neben Grafikdruck auch über die Möglichkeit des einfachen Textdrucks.

Menü FONT

Hier erscheinen die Schriftarten, die Sie auf Ihrer Diskette haben. Auch wenn Sie meinen, hier bereits alles zu kennen: Probieren Sie doch einmal LW_GREEK aus, und lassen Sie sich überraschen.

Menü STYLE

Hier finden sich neben den bekannten Möglichkeiten PLAIN TEXT, BOLD, ITALIC, OUTLINE und UNDERLINE zwei neue, nämlich SUPERSCRIPT und SUBSCRIPT.

SUPERSCRIPT ermöglicht Ihnen das Hochstellen von Zeichen, was etwa bei Anmerkungen sehr wichtig ist. SUBSCRIPT ermöglicht Ihnen dementsprechend das Tiefstellen. Darüber werden Sie z.B. froh sein, wenn Sie etwa chemische Formeln zu Papier bringen müssen. Wasser wird eben chemisch erst richtig als Wasser akzeptiert, wenn Sie mit GEOWRITE 2.0 mehr können als wir mit unserer Textverarbeitung: Sie erlaubt es uns nämlich nur, die Formel so wiederzugeben: Wasser = H_2O.

Probieren Sie das doch jetzt einmal mit GEOWRITE 2.0 und SUBSCRIPT aus. Und wenn Sie es sich besonders angenehm machen wollen, verwenden Sie noch den Kurzbefehl für SUBSCRIPT, nämlich die Verbindung von <Commodore>-Taste und spitzer Klammer links.

Die Formatierleiste

Kommen wir nun zur Formatierleiste, die vor allem für die optische Gestaltung des Textes zuständig ist. D.h., hier finden Sie vielfältige Möglichkeiten, Ihren Text speziell zu "layouten". Und im Vergleich zu früheren Versionen ist hier einiges hinzugekommen.

Linker und rechter Rand

Was Sie schon von GEOWRITE 1.2 und GEOWRITE 1.3. her kennen, ist die Möglichkeit, den linken und rechten Rand zu verändern. Dies können Sie erreichen, indem Sie eines der bei-

den "M" auf der Formatierleiste anklicken und dorthin setzen, wo Sie den Text in der Breite enden lassen wollen. Damit haben Sie vor allem die Möglichkeit, durch entsprechende Randwahl ständig den Text zu sehen und erst kurz vor dem Druck über die Bildschirmbreite hinauszuschieben.

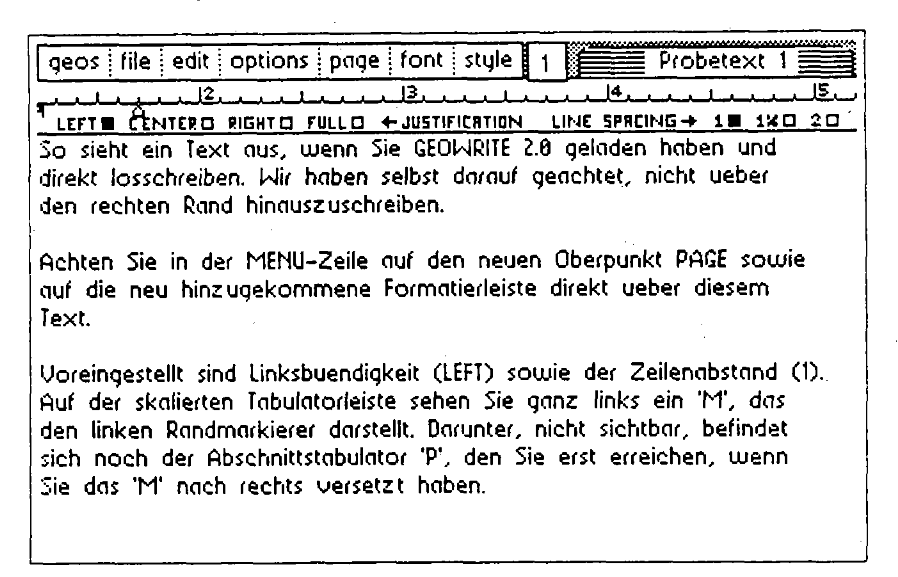


Bild 1.8.2.3: GEOWRITE 2.0-Grundeinstellung

Tabulatoren

GEOWRITE 2.0 bietet Ihnen auf derselben Leiste insgesamt acht Tabulatoren an, die wie kleine Dächer aussehen und die Sie an beliebigen Stellen anklicken können.

Wenn Sie zu Beginn einer Zeile die <Ctrl>-Taste und gleichzeitig <I> drücken, springt der Cursor zur nächsten Tabulatorposition. Neben diesen normalen und bekannten Tabulatoren bietet Ihnen GEOWRITE 2.0 noch zwei weitere Arten an:

1. den Abschnittstabulator: Er versteckt sich hinter einem "P" auf der Formatierleiste. Wenn Sie ihn anklicken, beginnt dort jeder neue Absatz, d.h., dieser wird eingerückt.

2. den Dezimaltabulator, einen schwarzen Kreis: Mit Dezimaltabulatoren kann man Zahlen formatiert untereinander schreiben. Dadurch steht jeweils ein Dezimalpunkt unter dem anderen.

Als erstes müssen wir einen Dezimaltabulator an der gewünschten Stelle setzen. Klicken Sie dazu auf der Tabulatorleiste an der gewünschten Stelle, und Sie erhalten einen normalen Tabulator. Drücken Sie nun die <Shift>-Taste, und er verwandelt sich in einen schwarzen Dezimaltabulator. Durch erneutes Drücken der <Shift>-Taste können Sie ihn wieder zurückverwandeln.

Um nun dort Zahlen untereinander zu positionieren, müssen Sie jeweils mit <Ctrl> plus <I> den Cursor auf die Tabulatorposition setzen und die Zahlen eingeben. Denken Sie dabei daran, daß statt des deutschen Kommas der amerikanische Punkt erwartet wird.

Jede Zahleneingabe beenden Sie bitte mit <Return> und setzen mit <Ctrl> plus <I> den Cursor erneut auf die Tabulatorposition. Das sieht dann etwa so aus:

```
   512.56
     4.70
  1000.00
    80.65
usw.
```

Sollten Sie wie wir an dieser Stelle Schwierigkeiten erhalten, weil GEOWRITE 2.0 nur zwei Ziffern vor dem Punkt anzeigt, so können Sie das Problem folgendermaßen lösen:

Schreiben Sie die Zahlen einfach am linken Rand untereinander. Wenn Sie mit allen Zahlen fertig sind, setzen Sie den Cursor jeweils vor die erste Ziffer an den Anfang der Zeile, und drücken Sie dann <Ctrl> + <I>. Dadurch "springen" die Zahlen dann in die richtige Position. Zum Abschluß dieses Teils der Benutzung der Formatierleiste noch ein wichtiger Hinweis. Da Sie mehrere verschiedene Einstellmöglichkeiten haben, müssen Sie immer damit rechnen, daß sich hinter einer Einstellung noch eine andere verbirgt.

Suchen Sie also nicht mit den Augen nach Ihrem vierten Normaltabulator, er könnte sich hinter dem Dezimalstellentabulator verbergen und durch dessen Verschieben leicht zu finden sein.

Anders ist das nun einmal nicht zu machen, wenn Sie nicht die Hälfte des Bildschirms mit speziellen Formatierleisten gefüllt haben wollen. Sie werden sich schnell daran gewöhnen.

Die nächsten vier Formatiermöglichkeiten sind zusammengefaßt unter dem Begriff "Justification". Übersetzt werden könnte es etwa mit "Justierung" (des Textes). Was damit genau gemeint ist, werden Sie am besten verstehen, wenn Sie sich auf die folgenden vier Möglichkeiten einmal einlassen und sie ausprobieren.

LEFT

Ist das zugehörige Kästchen schwarz, d.h., aktiviert, beginnt jede Zeile linksbündig, d.h., am linken Rand. Das ist keineswegs aufregend, aber als Normalfall muß es eben anwählbar sein.

CENTER

Nicht "normal" im Sinne von "gewöhnlich" schreiben Sie, wenn Sie dieses Kästchen aktiviert haben. Denn dann wird Ihr Text, je nach Ihrer Randeinstellung, um seine Mitte zentriert.

RIGHT

Mit diesem Kästchen erreichen Sie das Gegenstück zur Linksbündigkeit. Das heißt, GEOWRITE schreibt von rechts nach links. (Schauen Sie sich diesen interessanten Effekt einmal an!) Als Ergebnis stehen nun am linken Rand die Zeilenanfänge nicht mehr direkt untereinander.

FULL

Klicken Sie dieses Kästchen an, haben Sie einen Randausgleich (Blocksatz). Das heißt, Sie schreiben links- und rechtsbündig, wie Sie es von professionellen Druckwerken gewöhnt sind.

LINE SPACING

Mit diesem Begriff ist der Zeilenabstand gemeint. Sie können hier also auf dem Bildschirm das erreichen, was Sie auch an einer Schreibmaschine (mit denselben Werten) einstellen können. Wählen können Sie zwischen einfachem (1), anderthalbfachem (1 1/2) und zweifachem (2) Zeilenabstand.

Eine außergewöhnliche Fähigkeit von GEOWRITE 2.0 besteht wohl darin, daß man abschnittsweise formatieren kann, womit sich eine Fülle von Layout-Möglichkeiten ergeben. So können Sie beispielsweise die Überschrift zentrieren, die ersten zwei Abschnitte einrücken, dann ein wenig rechtsbündig schreiben und anschließend für den Rest Ihres Schreibens zum allgemein üblichen Blocksatz übergehen, bevor Sie am Ende dem Empfänger die Rechnung mit Hilfe der Dezimaltabulatoren präsentieren.

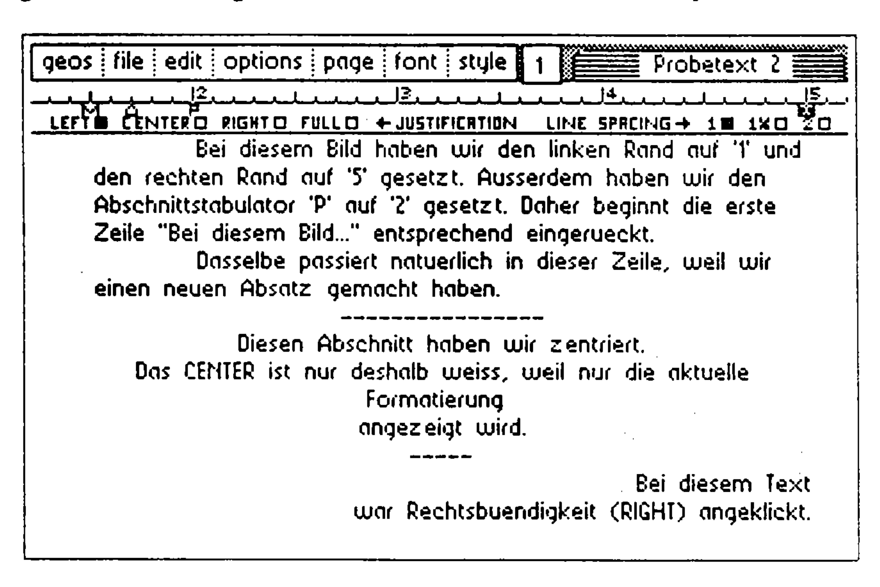


Bild 1.8.2.4: Layout mit GEOWRITE 2.0

Unsere Gesamteinschätzung von GEOWRITE 2.0:

Insgesamt zeigt sich GEOWRITE in der neuen Version 2.0 deutlich verbessert. Gefallen haben uns vor allem die neuen Bewe-

gungsmöglichkeiten mit den Cursor-Tasten. Durch den rechten Randausgleich (Blocksatz), die Zentriermöglichkeit und die Suchen/Ersetzen-Funktion ist professionelles Arbeiten möglich.

Sollten Sie zu denen gehören, die als erste in die neue Laserdrucktechnik einsteigen, werden Sie auch hier GEOWRITE 2.0 zu schätzen wissen. (Oder kennen Sie ein anderes Programm, das es ermöglicht, mit dem C64 einen Laserdrucker anzusteuern?)

Noch verbessert werden könnte unserer Meinung nach die Schreibgeschwindigkeit. Dazu gehört beispielsweise die Schnelligkeit der REPEAT-Funktion (Dauerdruck von Buchstaben).

Tabelle der Tastatur-Kurzbefehle für GEOWRITE 2.0

Einige der folgenden Kurz-Tastaturbefehle sind Ihnen schon von GEOWRITE 1.3 her bekannt. Wir haben sie der Vollständigkeit halber noch einmal aufgeführt.

EDIT-Menü

<Commodore>-Taste plus <X> = CUT
<Commodore>-Taste plus <C> = COPY
<Commodore>-Taste plus <T> = PASTE (TEXT)
<Commodore>-Taste plus <W> = PASTE (PICTURE)

OPTIONS-Menü

<Commodore>-Taste plus <S> = SEARCH
<Commodore>-Taste plus <N> = FIND NEXT
<Commodore>-Taste plus <Y> = CHANGE, THEN FIND
<Commodore>-Taste plus <H> = OPEN HEADER
<Commodore>-Taste plus <F> = OPEN FOOTER
<Commodore>-Taste plus <V> = SELECT PAGE

PAGE-Menü

<Commodore>-Taste plus <Pfeil nach links > = PREVIOUS PAGE
<Commodore>-Taste plus <+> = NEXT PAGE
<Commodore>-Taste plus <G> = GOTO PAGE
<Commodore>-Taste plus <L> = PAGE BREAK

STYLE-Menü

<Commodore>-Taste plus <P> = PLAIN TEXT
<Commodore>-Taste plus <B> = BOLD
<Commodore>-Taste plus <I> = ITALIC
<Commodore>-Taste plus <O> = OUTLINE
<Commodore>-Taste plus <U> = UNDERLINE
<Commodore>-Taste plus <spitze Klammer rechts> = SUPERSCRIPT
<Commodore>-Taste plus <spitze Klammer links> = SUBSCRIPT

Befehlszeile oben auf jedem Blatt

<Commodore>-Taste plus <A> = LEFT JUSTIFY
<Commodore>-Taste plus <E> = CENTER JUSTIFY
<Commodore>-Taste plus <R> = RIGHT JUSTIFY
<Commodore>-Taste plus <J> = FULL JUSTIFY
<Commodore>-Taste plus <K> = SINGLE SPACE
<Commodore>-Taste plus <M> = ONE AND A HALF SPACE
<Commodore>-Taste plus <D> = DOUBLE SPACE

1.8.3 Serienbriefe

Im folgenden wollen wir Ihnen nun die verschiedenen Möglichkeiten demonstrieren, wie man mit Hilfe von GEOWRITE und GEOMERGE Serienbriefe herstellen kann. Dabei werden wir insgesamt drei grundsätzliche Möglichkeiten kennenlernen.

Bei der ersten erstellen Sie zunächst den Brief und geben dann jeweils zum Drucken die Adresse ein, an die Sie den Brief geschickt haben möchten.

Serienbriefe mit fortlaufender Eingabe der Adresse

Wir wählen also jetzt zunächst GEOWRITE an und schreiben einen kurzen Brief. Solche Texte, die später an verschiedene Adressaten in Serie gehen sollen, nennen wir hier jetzt ab sofort "FORMBRIEF". Als Namen für unseren ersten und sehr kurzen FORMBRIEF wählen wir hier "formbrief 1".

Wir schreiben jetzt also unseren "formbrief 1": Und bitte, übernehmen Sie das, wenn Sie sich noch nicht sicher fühlen, bis auf die Klammern genauso!

```
»firstNames«,ich mag Dich wirklich sehr!
```

Anschließend verlassen wir GEOWRITE und klicken Geomerge an. Sollten Sie GEOMERGE jetzt erstmals benutzen, müssen Sie es genauso wie GEODEX installieren.

Nachdem GEOMERGE geladen ist, erscheint ein Fenster, das uns an das entsprechende Fenster von GEOWRITE erinnert und uns die Auswahl von Briefen ermöglicht; nennen wir es daher GEOMERGE-Auswahlfenster. Wir klicken unseren Beispieltext "formbrief 1" und anschließend ÖFFNEN (OPEN) an.

Es erscheint ein zweites Fenster, das uns fragt: "Use geowrite file for merge information?", also ob wir für die Serienbriefinformation ein mit GEOWRITE erstelltes File verwenden wollen. Dieses müßte dann die Adresse enthalten. Auf diese Möglichkeit gehen wir weiter unten ein.

Da wir noch keine entsprechende GEOWRITE-Adressensammlung haben, klicken wir jetzt "no" an und geraten in ein drittes Fenster, das uns verschiedene Druckmöglichkeiten anbietet.

In der obersten Zeile können wir wählen zwischen "High", "Draft" und "NLQ", dabei ist "High" vorbesetzt. Wir wählen jetzt, weil das am schnellsten geht, durch Anklicken "Draft" an.

("NLQ" nutzt, falls vorhanden, die "Near Letter Quality"-Eigenschaften Ihres Druckers aus, "High" ergibt Grafikdruck. Außerdem können wir in der nächsten Zeile die Seite angeben, von der und bis zu der gedruckt werden soll ("from page to page"). Beide Möglichkeiten sind mit "1" vorbesetzt, und wir lassen diese Einstellung auch bestehen.

In der nächsten Zeile können wir wählen zwischen "single sheet" = Einzelblatt und "tractor feet" = Endlospapier. Anschließend klicken wir das OK in der untersten Zeile an. Es erscheint ein weiteres Fenster mit der Frage : "Enter data to substitute for label "firstNames":".

Damit deutet das Programm an, daß es sich jetzt die Anschriften, an die der Serienbrief geschickt werden soll, über die manuelle Eingabe mit Hilfe der Tastatur holen will. Und zwar geht es bei unserem "formbrief 1" nur um Vornamen, sonst würde der Text wenig Sinn ergeben. Wenn Sie es nicht glauben, so geben Sie ruhig Vor- und Familiennamen direkt aufeinander folgend ein, Sie werden schon sehen, was dabei herauskommt.

Wenn Sie hier jetzt einen Vornamen eingeben, etwa "Peter", so bekommen Sie gleich unseren Serienbrief entsprechend ausgedruckt, da in unserem "formbrief 1" nur ein label (= Platzhalter) zu ersetzen war. Es steht dann da:

"Peter, ich mag dich wirklich sehr."

Ob Sie Ihre Zuneigung nun in Serie gehen lassen möchten, können Sie entscheiden, wenn Sie die nächste Frage: "Do another merge of this document" mit "Yes" beantworten. Dann können Sie nämlich einen neuen Namen eingeben, und ein zweiter Serienbrief wird gedruckt. Unten werden übrigens die gedruckten Briefe gezählt. Oben links in der Ecke hinter der Bezeichnung "document:" finden Sie auch immer zur Kontrolle den Namen des Textes, den Sie in Serie gehen lassen.

Nachdem wir an diesem einfachen Beispiel das grundsätzliche Verfahren kennengelernt haben, mit dem man Serienbriefe erstellen kann, wenden wir uns nun einer zweiten Möglichkeit zu, die schon sehr viel professioneller und wirklichkeitsnäher ist.

Serienbriefe mit Hilfe von GEODEX

Stellen wir uns nun einmal vor, Sie möchten Freunde zu einer Geburtstagsfeier einladen und sich dabei von GEOS unterstützen lassen. Zu diesem Zweck laden Sie zunächst GEODEX und gehen auf den zweituntersten Punkt in der Menüleiste. Es erscheint dann ein Fenster, das Sie auffordert, einen Namen für eine Serienbrief-Adreßsammlung einzugeben ("enter merge file name"). Wir geben unserem Beispiel einfach den Namen "adressen".

Damit stehen alle Einträge der GEODEX-Kartei jetzt für einen Serienbrief zur Verfügung. Um diesen nun schreiben zu können, verlassen wir GEODEX und laden GEOWRITE. Als Namen für unseren zweiten Formbrief wählen wir "geburtstag". Nun kann es losgehen. Als Text geben wir ein (achten Sie auf die Klammern):

```
»first« »last«
»addr«
»city«


Lieber »first«

Ich möchte Dich gerne zu meinem Geburtstag am 3.4.87 einladen.
Hoffentlich kommst Du. Ich würde mich sehr freuen.

Dein Gerd
```

Anschließend verlassen wir GEOWRITE mit QUIT. Nachdem wir GEOMERGE wieder geladen haben, taucht ein Fenster auf, das uns auffordert, den Namen des Textes einzugeben, den wir als Serienbrief ausdrucken lassen wollen (Choose form document). Wir wählen natürlich jetzt unseren soeben geschriebenen Formbrief "geburtstag". Oben links zeigt uns das Programm auch den gewählten Text.

Anschließend fragt es uns, ob wir ein GEOMERGE-File verwenden wollen. Diesmal können wir das bejahen. Im folgenden Fenster wählen wir als Adreßkartei ("merge file") die Datei, der wir vor kurzem den Namen "adressen" gegeben haben.

Anschließend zeigt uns GEOMERGE wieder das schon bekannte Print Menü, in dem wir uns wieder für Draft (schnelle Schrift) entscheiden. Sobald wir OK angeklickt haben, fängt der Drucker an zu arbeiten. Bei einer entsprechenden Datei sieht der erste Brief dann so aus:

```
Jürgen Becker
Meierskamp 14
4430 Bochum
...
```

```
...
Lieber Jürgen

Ich möchte Dich gerne zu meinem Geburtstag am 3.4.87 einladen.
Hoffentlich kommst Du. Ich würde mich sehr freuen
Dein Gerd
```

Nachdem nun alle Adreßkarten verwendet worden sind, fragen wir uns, ob es wirklich richtig war, alle Leute, deren Daten wir in unserem Karteikasten haben, auch zu unserer Geburtstagsfeier einzuladen. Da unsere Kellerbar höchstens Platz für etwa zwanzig Leute hat, überlegen wir uns, ob man nicht gezielter auswählen sollte.

Nun können wir natürlich alle Adressen mit den entsprechenden Briefen ausdrucken lassen, die nicht benötigten in den Papierkorb werfen und schon jemanden losschicken, um für den kommenden Geburtstag der Schwester mehrere Kilo Endlospapier kaufen zu lassen.

Doch halt, da fällt uns ein, daß es doch eine Möglichkeit gab, die gesamte Kartei mit Hilfe der "group"-Felder zu Gruppen zusammenzufassen.

Das bedeutet also, daß wir aus der Gesamtkartei diejenigen heraussuchen, die wir zu unserem Geburtstag einladen möchten. Und bei Ihnen geben wir ins erste "group"-Feld die drei Buchstaben "pri" ein. Bei all den Adressen, die für unsere Lieblingsschwester Gabi interessant sind, geben wir als Abkürzung "gab" ein.

Und nun können wir neue Serienbrief-Adreßkarteien erstellen ("geb pri" und "geb gabi"), die wir jetzt jetzt entsprechend auch ausdrucken lassen können.

Am Schluß haben wir einen Stapel von zum Beispiel 19 Briefen, die unsere Freunde zu unserem Geburtstag einladen, und einen Stapel von meinetwegen 17 Briefen, die dasselbe für unsere Lieblingsschwester Gabi leisten.

Kommen wir nun zur dritten Möglichkeit, Serienbriefe zu erstellen, wobei neben GEOMERGE vor allem GEOWRITE benutzt wird, d.h., mit diesem Schreibprogramm wird nicht nur der Text des Serienbriefes erstellt, nein, wir erstellen, man höre und staune, mit GEOWRITE auch die spezielle Serienbrief-Adreßkartei der Leute, an die wir diesen Brief schicken wollen.

Serienbriefe mit Hilfe von GEOWRITE-Adreßkarteien

Um die Voraussetzungen für diese Möglichkeit zu verstehen, müssen wir uns zunächst einmal ansehen, wie ein "Geomerge-File", d.h. also eine Serienbrief-Adreßkartei aussieht. Wenn man sie sich mit GEOWRITE anguckt (tatsächlich, das geht), so stehen auf der ersten Seite untereinander die Labels, d.h., die Platzhalter first, middle, last, addr, city, state, zip, misc, phone, und am Schluß steht ein Sternchen.

Auf den nächsten Seiten folgt jeweils eine Karteikarte des Merge-Feldes, in unserem Fall der Datei "adressen", die wir uns gerade anschauen, durch <Return> getrennt. Sollten Sie sich gerade eine Ihrer fortschrittlichen "Geburtstags-Specials", d.h. mit "group" sortierten Karteien ansehen (etwa "pri" oder "gab"), dann finden Sie immer dort eine leere Seite, wo in Ihrer Gesamtdatei eine Karte war, die die "group"-Bedingung nicht erfüllt. Wichtig ist, daß diese leere Seite nicht ausgedruckt wird, so daß Sie wirklich eine Menge Endlospapier sparen. Probieren wir das doch einmal aus:

Zunächst laden wir GEOWRITE. Den neuen Text nennen wir, weil es eine mit GEOWRITE erstellte Kartei ist, "write kartei". Auf der ersten Seite geben wir einfach die allgemeinen "labels", also die Platzhalter ein, die später durch die konkreten Angaben/Daten ersetzt werden sollen. Also z.B.:

```
Vorname
Nachname
Postleitzahl
Ort
Straße
Telefon
*
```

Das Sternchen am Schluß zeigt GEOMERGE an, daß an dieser Stelle Schluß ist. Achten Sie darauf, daß Sie hinter jedem "label" bitte nur <Return> drücken und hinter dem Sternchen PAGE BREAK wählen, um eine neue Seite zu erhalten. Auf der zweiten Seite geben wir dann die Adressen in genau derselben Reihenfolge ein, z.B.:

```
Walter
Meier
3467
Neubrück
Waldauerstr. 15
02564/456
*
```

So könnte man auf den nächsten Seiten fortfahren. Dieses mit Hilfe von GEOWRITE erstellte Merge-File "write kartei" würde schließlich von GEOMERGE in derselben Weise zu Serienbriefen verarbeitet wie in dem Beispiel oben mit GEODEX.

Der Vorteil ist hier natürlich, daß man viel flexibler ist (z.B. Eingabe der deutschen Platzhalter (=label) oder auch Eingabe von viel mehr und ganz anderen Platzhaltern usw.).

Bei dem entsprechenden Fenster in GEOMERGE beantworten wir die entsprechende Frage, ob wir ein "GEOWRITE-File" benutzen möchten, mit "YES", und schon können wir unsere neue Kartei "write kartei" benutzen, um Serienbriefe mit z.B. unserem Formbrief "geburtstag" zu benutzen. Kommen wir nun noch zu einer weiteren Möglichkeit, unsere Serienbriefe zu verbessern. Wenn Sie in Ihrer Kartei männliche und weibliche Adressaten gespeichert haben, was wohl die Regel ist, müssen Sie befürchten, in einem der Serienbriefe die folgende Anrede vorzufinden.

```
Lieber Beate,

(...)
```

Diese ärgerliche Geschlechtsumwandlung einer lieben Freundin können Sie vermeiden, indem Sie in Ihren Formbrief ein entsprechendes Auswahlkommando eingeben. Grundsätzlich gibt es hierfür den folgenden Befehl:

```
»IF labelName="value"«text1»ELSE«text2»ENDIF«
```

Damit ist folgendes gemeint: Wenn das Feld "labelName" den Eintrag "value" enthält, wird "text1" gedruckt, im anderen Fall "text2". Die Schlußklammer gibt nur das Ende des Vergleichsbefehls an.

Wir wollen das einmal bei unserem Beispiel benutzen: Verändern wir also das Rundschreiben zu unserem Geburtstag in folgender Weise: Bitte laden Sie GEOWRITE (1.2/1.3/2.0), und erstellen Sie zuerst den eigentlichen Brief für das Schreiben mit dem Namen "Geb.Form":

```
»Vorname« »Nachname«
»Strasse«
»Ort«


»IF Geschlecht = "m"«Lieber»ELSE«Liebe»ENDIF« »Vorname«
Ich möchte Dich gerne zu meinem Geburtstag am 3.4.87 einladen.
Hoffentlich kommst Du. Ich würde mich freuen.
                                          Dein Gerd
```

```
geos | file | edit | options | page | font | style | 1 |  Geb.Form
LEFT ■ CENTER □ RIGHT □ FULL □ ← JUSTIFICATION   LINE SPACING → 1 ■ 1½ □ 2 □
<<Vorname>> <<Nachname>>
<<Strasse>>
<<Ort>>


<<IF Geschlecht = "m">>Lieber<<ELSE>>Liebe<<ENDIF>> <<Vorname>>
Ich moechte Dich gerne zu meinem Geburtstag am 3.4.87 einladen.
Hoffentlich kommst Du. Ich wuerde mich freuen.

                                          Dein Gerd
```

Bild 1.8.3.1: Serienbrief mit Hilfe von GEOWRITE-Formbrief

Wenn Sie mehrere der späteren Formbriefe auf einer Seite haben möchten, so beenden Sie die Arbeit an "Geb.Form" mit SCHLIESSEN (CLOSE), andernfalls setzen Sie am Schluß wie in unserer Abbildung noch einen Seitenumbruch.

Nun benötigen wir noch einen GEOWRITE-Text, der die Labels und die Adressen enthält. Daher erstellen wir einen zweiten Brief mit dem Namen "Geb.Daten". Auf der ersten Seite schreiben wir:

```
Vorname
Nachname
Postleitzahl
Ort
Strasse
Telefon
Geschlecht
*
```

```
geos | file | edit | options | page | font | style | 1 | Geb.Daten
|2 |3 |4 |5
LEFT ■ CENTER □ RIGHT □ FULL □ ← JUSTIFICATION   LINE SPACING → 1 ■ 1½ □ 2 □
Vorname
Nachname
Postleitzahl
Ort
Strasse
Telefon
Geschlecht
*
```

Bild 1.8.3.2: Serienbrief mit GEOWRITE - Liste der Labels

Hinter dem "*" fügen Sie dann bitte mit <Commodore> + <L> (bzw. mit SEITENUMBRUCH (PAGE BREAK)) einen Seitenumbruch ein, und auf der nächsten Seite beginnt dann unser erster "Datensatz":

```
Gerd
Schmidt
4400
Münster
Kolbeweg 13
0251-147887
m
*
```

Bitte fügen Sie in der Zeile unter dem "*" wieder den nötigen Seitenumbruch ein, und beachten Sie das "m" in der Zeile darüber. Mit diesem Label wird später entschieden, ob es "Lieber" oder "Liebe" heißen wird. Nun wollen wir zum ausprobrieren noch eine zweite Adresse eintragen:

```
Petra
Mayer
4000
Düsseldorf
Rubensstraße 1a
0211-4512919
w
*
```

Nun können wir GEOWRITE verlassen und den neuen, vollendeten Serienbrief an die beiden Gäste drucken lassen. Laden Sie GEOMERGE, wählen Sie "Geb.Form" und anschließend "YES" und "Geb.Daten".

1.8.4 DESK PACK 1

Wenn Sie mit uns zu denen gehören, die sich nicht nur an GEOS als praktische Benutzeroberfläche für den C64 gewöhnt haben, sondern sich auf Erweiterungen des Angebots freuen, sollten wir uns zusammen einmal ein erstes Zusatzpaket zur GEOS-Grundausrüstung ansehen, das von Berkeley Softworks unter dem Namen DESK PACK *1 entwickelt worden ist.*

Umfang des Paketes

Neben DESKTOP in der Version 1.3 befinden sich vier weitere Programme auf der Diskette.

Dabei handelt es sich um einen Kalender (CALENDAR), ein Programm zur Erstellung von Icons (ICON EDITOR), ein Programm, mit dem Sie Grafiken einiger anderer Programme ins GEOS-Format umwandeln können (GRAPHICS GRABBER) und schließlich ein Accessory, das mit seinem verheißungsvollen Namen "BLACK JACK", über die Welt der harten Büro- und Schreibtischarbeit hinausreicht.

Installierung

Alle Programme auf dieser Diskette müssen zunächst installiert werden, d.h., sie werden mit der GEOS-Version gestartet, die Sie ständig benutzen wollen und von der sie anschließend nicht mehr getrennt werden können. Nähere Informationen dazu finden Sie in dem Abschnitt, in dem die GEOS-Version 1.3 vorgestellt wird.

Der Kalender

Bei der Vorstellung des ersten Accessories (CALENDAR) gehen wir vom 9. April 1987 als aktuellem Tag aus. Nach dem Laden des Kalenders durch zweimaliges Anklicken sehen wir zunächst das Eröffnungsbild. Ganz oben links haben wir ein kleines Menü mit nur zwei Punkten. Zu FILE findet sich als Unterpunkt QUIT, bei CHANGE haben wir die Möglichkeiten, MONTH, YEAR und TO PRESENT anzuwählen.

Im eigentlichen Kalenderfenster findet sich oben rechts das bekannte Schließsymbol, das bei uns aber nicht immer funktioniert hat. Sicherer verlassen Sie den Kalender wieder durch Anwählen des oben erwähnten Menüfeldes QUIT unter FILE.

In der ersten Zeile des eigentlichen Kalenderblattes finden Sie die englischen Abkürzungen für die Wochentage

```
S: Sunday/Sonntag
M: Monday/Montag
T: Tuesday/Dienstag
W: Wednesday/Mittwoch
T: Thursday/Donnerstag
F: Friday/Freitag
S: Saturday/Samstag bzw. Sonnabend
```

Darunter erscheinen dann die Tage des aktuellen und im Voreinsteller (PREFERENCE MANAGER) eingestellten Monats in der richtigen Anordnung. Unter einer dreifachen Querlinie werden Monat und Jahr noch einmal angezeigt. Ganz rechts auf dem Monatskalenderblatt finden Sie über der Dreifachlinie das bekannte Eselsohr und unter der Linie ein Fragezeichen.

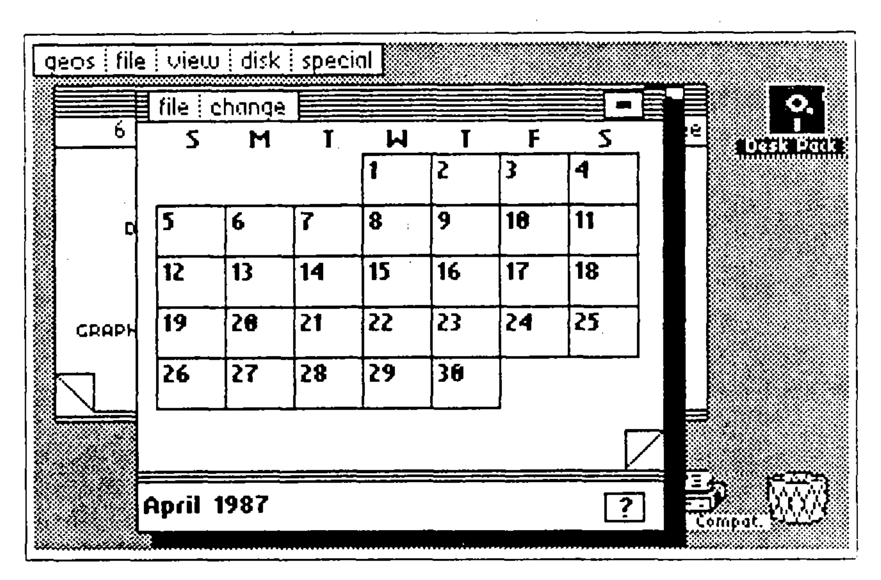


Bild 1.8.4.1: Kalenderblatt des laufenden Monats

Probieren wir nun den Kalender erst einmal aus. Er wird sich dabei als überaus fähig erweisen. Zu diesem Zwecke gehen wir auf CHANGE und klicken YEAR an. Jetzt können wir eine vierstellige Zahl eingeben und damit ein beliebiges Jahr zwischen 0000 und 9999 wählen. Probieren Sie es doch einmal mit Ihrem Geburtsjahr aus. Wenn Sie dann noch über MONTH den richtigen Monat anklicken, erscheint innerhalb kürzester Zeit der Monat, der Ihren Eltern eine besondere Freude gemacht hat. Sie können jetzt schnell nachschauen, ob Sie etwa ein Sonntagskind sind. Sollte das nicht der Fall sein, beweisen Sie dem Schicksal das Gegenteil! Auf jeden Fall können Sie mit Hilfe dieses Kalenders erstaunlich schnell feststellen, welcher Wochentag sich hinter einem beliebigen historischen Datum verbirgt.

Probieren wir es doch einmal mit einem Datum aus, das Sie wohl noch aus der Schule oder aber von Ihrem letzten Parisurlaub her kennen. Am 14. Juli jeden Jahres feiern die Franzosen den Sturm auf die Bastille. Wenn Sie unter CHANGE/YEAR das Jahr 1789 verlangen und anschließend den Monat Juli wählen, wissen Sie sehr schnell, daß das berühmte Ereignis an einem ... Na, probieren Sie es ruhig selbst aus.

Ein zweites Beispiel möchten wir Ihnen noch demonstrieren, weil es uns einfach zu sehr in den Fingern gejuckt hat. Wir sind einfach einmal davon ausgegangen, daß Christi Geburt wirklich im Jahre 0 stattgefunden hat (die Historiker und Theologen mögen unsere Unbedarftheit verzeihen). Und mit Hilfe unseres Kalenders haben wir festgestellt, daß der erste "Weihnachtstag" ein ... war.

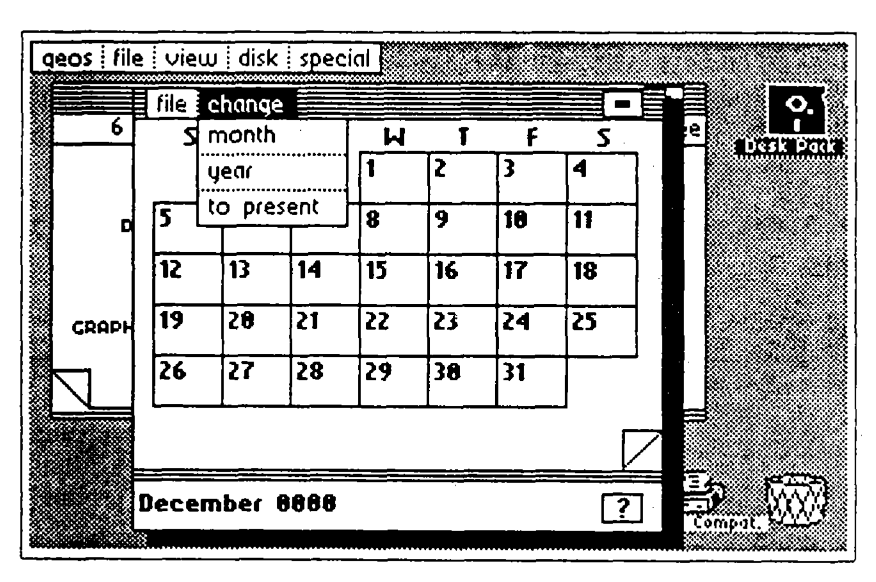


Bild 1.8.4.2: Beispiel für die Nutzung des "ewigen" Kalenders

Nun aber genug dieser (wenn auch entzückenden) Spielereien. Fangen wir an, unseren Kalender für die Gegenwart zu verwenden. Wir möchten z.B. einige Einträge für den 9.4. machen.

Also klicken wir das Feld mit der 9 an. Ein leeres Fenster erscheint mit drei Anklickmöglichkeiten über der Dreifachlinie: CLEAR, OK und CANCEL. Außerdem befindet sich rechts unten noch ein Eselsohr. Unter der Dreifachlinie ganz links finden wir jetzt nicht nur den Monat, sondern auch den genauen Tag, den wir angewählt haben. In der amerikanischen Schreibweise, an die man sich schnell gewöhnt, sieht das dann so aus:

```
April 9, 1987
```

Für unsere Zwecke geben wir jetzt zum Beispiel ein:

```
0800 Große Inspektion (Auto Meier)
1015 Informatik-Klausur
1300 Treffen mit Gabi in der Mensa
1615 Arbeitsgruppe PASCAL
2000 "Im Namen der Rose" (Apollo-Kino)
```

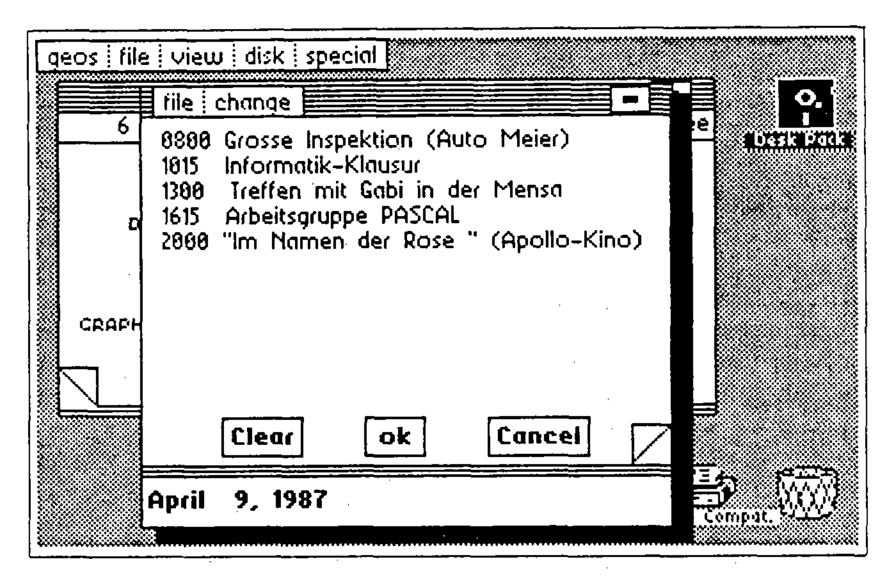


Bild 1.8.4.3: Beispiel für das Kalenderblatt eines Tages

Anschließend gehen wir auf OK. Die Daten werden dann auf der Diskette abgespeichert. Mit CLEAR könnte man das gesamte Blatt wieder löschen, und mit CANCEL kann man wie immer den gesamten Vorgang abbrechen und zum Grundbild des Ka-

lenders zurückkehren. Dasselbe geschieht übrigens, wenn man auf dem Kalenderblatt eins der beiden Eselsohren rechts unten anklickt.

Wenn Sie die Daten unter dem 9. April eingetragen haben und ins Monatsgrundbild zurückgekehrt sind, können Sie an einem Sternchen erkennen, daß zu einem bestimmten Tag eine Eintragung vorhanden ist. So behält man leichter den Überblick. In unserem Fall befindet sich das Sternchen im Feld mit der 9.

Notieren wir uns nun zu zwei weiteren Tagen etwas. Uns ist eben die Benachrichtung unseres fürsorglichen Zahnarztes zu Gesicht gekommen. Da wir gerade dessen Termine gerne und schnell vergessen, wählen wir über CHANGE und MONTH den Mai 1987 an und vermerken dort ebenfalls unter dem 9:

```
0915 Zahnarzt.
```

Endgültig verbucht wird die Eintragung wieder durch Anklicken von OK. Als nächstes wollen wir einmal überprüfen, wie der Kalender auf Daten reagiert, die vor dem heutigen Tag liegen. Wir klicken unter CHANGE und MONTH den Januar an und geben dort für den 1. Januar ein: 08.00 Seminar.

(An dieser sicher ungewöhnlichen Verwendung des Neujahrstages mögen Sie zum einen unseren Fleiß, zum anderen die Unbekümmertheit erkennen, mit der GEOS alles akzeptiert.) Nachdem wir den Eintrag mit OK abgespeichert haben, wählen wir einen kürzeren Weg zurück in die Gegenwart, und zwar unter dem Menüpunkt CHANGE durch Anklicken von TO PRESENT.

Wenn wir jetzt im Monatsgrundbild das Fragezeichen unten rechts anklicken, nennt das Programm uns in der zeitlich richtigen Reihenfolge alle Tage, zu denen Einträge vorhanden sind. In unserem Falle also:

```
Januar 1, 1987
April 9, 1987
Mai 9, 1987.
```

Auf einem solchen Blatt würden uns maximal 20 Einträge angezeigt. Mit Hilfe des Eselsohrs kann man dann vor- bzw. zurückblättern.

Möchte man nun die Einträge zu einem der Tage sehen, braucht man ihn nur in dieser Übersicht anzuklicken, und anschließend erscheint der entsprechende Tag mit seinen Kalendernotizen.

Verlassen wir nun zunächst einmal den Kalender, der Ihnen hoffentlich in Zukunft eine große Hilfe ist. Dazu wählen wir unter FILE QUIT an und geraten so wieder dorthin, von wo wir ausgegangen sind, etwa zum DESKTOP. Ebenso wäre es natürlich auch möglich, das Accessory CALENDAR aus GEOWRITE heraus anzuwählen und hinterher auch dorthin zurückzukehren.

ICON EDITOR

Mit dem ICON EDITOR bietet Ihnen Berkeley Softworks nun auch eine Möglichkeit, Files mit einem individuellen ICON zu versehen. Dies ist besonders interessant bei den Files, die nur mit einem ziemlich ausdruckslosen und allgemeinen Standard-Icon auf dem Bildschirm erscheinen, etwa alle selbst erstellten BASIC- oder auch Assembler-Programme.

Installation und Einstieg

Wir gehen davon aus, daß Sie auch den ICON EDITOR so installiert haben, wie wir es bei der Vorstellung von GEOS V1.3 beschrieben haben. Nach dem Laden erscheint zunächst ein Eingangsbild, wo Sie entsprechend ein File wählen können, das Sie mit einem ICON versehen wollen (File-Select-Box).

Werden nicht alle Files, die Sie auf der Diskette haben, im Fenster angezeigt, können Sie hoch- und runterscrollen. Wichtig ist, daß es in diesem Eingangsbild ein "Disk-Feld" gibt, das Ihnen erlaubt, jederzeit die Diskette zu wechseln.

In unserem Falle wählen wir ein Programm, das noch nicht ins GEOS-Format gebracht worden ist. Nach dem Anklicken und dem Öffnen mit "OPEN" erscheint ein neues Fenster mit dem

schon erwarteten Hinweis: "This File is not a Geos File. Convert to Geos File Type?". Das heißt, wir werden jetzt gefragt, ob wir unser File ins GEOS-Format umwandeln wollen. Nach dem Anklicken von "YES" wird es umgewandelt. Dazu müssen noch einige Angaben gemacht werden. Ein neues Fenster erscheint, auf dem nach dem Typ unseres Files gefragt wird: "What Type of File is this?". Drei Möglichkeiten zur Wahl werden uns angeboten: Links "BASIC", dann "ASM" für "Assembler" und ganz rechts "DATA" für "Daten-Files".

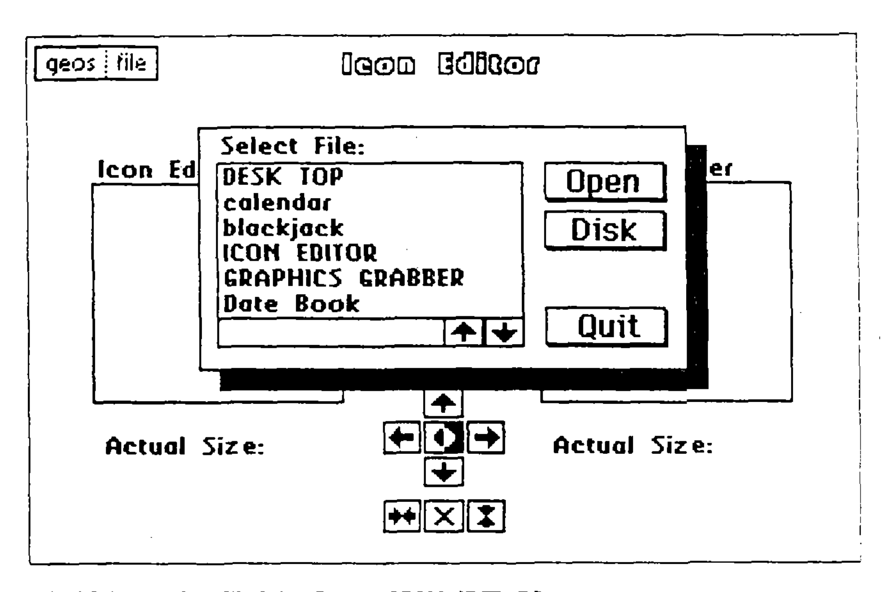


Bild 1.8.4.4: Die File-Select-Box des ICON EDITORS

Was hier schon deutlich wird, ist, daß wir mit dem ICON EDITOR nicht den File-Typ eines Accessories oder einer Application wählen können. Das geht nur mit dem FILEMASTER, den Sie ja weiter vorne in diesem Buch finden.

Wir wählen jetzt BASIC an. Ein neues Fenster fragt uns nach dem "PERMANENT NAME, d.h. nach dem Namen, der später auch immer im INFO-Bildschirm auftaucht. Hier können Sie z.B. Ihre eigene Entwicklungsnummer des Programms mit angeben (z.B. "HARDCOPY 1.3"). Dieser hier einzugebende Name ist nicht der, der später unter dem ICON erscheint.

Ein neues Eingabefenster verlangt entsprechend dem INFO-Bildschirmaufbau den Namen des Autors des Programms ("Enter Name of Author!"). Wir wählen jetzt als Beispiel "Werner".

Die Floppy arbeitet kurz, und anschließend erscheint das ICON-EDITOR-Grundbild, das folgendermaßen aussieht:

Ganz oben links haben wir eine Menüzeile, bestehend aus GEOS und FILE. Unter dem Menüpunkt GEOS finden wir das Info zum ICON EDITOR, außerdem wie üblich alle Accessories, da der ICON EDITOR ein Application ist.

Unter FILE haben wir "Save ICON", "Recover ICON", "Remove ICON" und "QUIT". Unter der Menüzeile findet sich der Name des Files, zu dem wir das ICON erzeugen wollen, und rechts davon wird noch einmal der Name der Diskette angezeigt, mit der wir arbeiten.

Darunter finden wir zwei große Quadrate. Über dem linken steht "ICON EDIT WINDOW" und rechts finden wir "ICON BUFFER". Das linke ist das eigentliche Arbeitsfeld, in dem man zeichnet. Rechts hat man die Möglichkeit, etwas kurzzeitig zu sichern.

Zwischen den beiden Quadraten finden sich zwei Pfeile. Der obere heißt "Copy to Buffer". Wenn man diesen Pfeil anklickt, kopiert man den Inhalt des linken Quadrates in das rechte. Der untere zeigt die umgekehrte Richtung an und heißt dementsprechend "Load from Buffer".

Darunter in der Mitte gibt es acht Anklickfelder. Fünf davon bilden ein Kreuz mit einem Mittelfeld. Jeweils an den Kreuzarmen haben wir die Möglichkeit, das Bild nach oben, unten, nach rechts und nach links zu scrollen. Das Feld in der Mitte ermöglicht uns das Invertieren des ICONS, an dem wir arbeiten.

Unter diesen fünf Feldern sind noch drei Anklickfelder horizontal angeordnet. Links kann man an der vertikalen Achse spiegeln. Das heißt, was vorher links war, erscheint dann rechts. Mit dem rechten Feld kann man an der horizontalen Achse spie-

geln, d.h. ein Bild auf den Kopf stellen. Und mit dem mittleren Feld kann man das Bild löschen. Damit stellt der ICON EDITOR komfortable Bearbeitungsmöglichkeiten zur Verfügung. Unter den beiden Arbeitsfenstern gibt es auch noch jeweils eine Angabe namens "Actual size". Hier ist im Augenblick noch nichts zu sehen, aber sobald wir anfangen zu zeichnen, würde das Ergebnis hier in Echt-Ausführung gezeigt, wie Sie es auch vom FILEMASTER her schon kennen.

Um nun mit dem Erstellen eines ICONS zu beginnen, fahren wir mit der Maus in das links Arbeitsfenster. Sofort geht es langsamer, und der Pfeil wird zu einem hellen kleinen Quadrat, wie man es auch vom Einzelpunktmodus (PIXEL EDIT) in GEOPAINT kennt. Auch das Setzen und Löschen von Punkten geschieht wie im entsprechenden Modus bei GEOPAINT: Durch einmaliges Klicken kommt man in den Malmodus, kann dann entsprechend so lange Striche ziehen, wie man will. Klickt man noch einmal, hebt man den Schreibmodus auf und hat den Zeichenstift gewissermaßen vom Papier abgehoben. Klickt man an einer Stelle, wo schon ein Punkt ist, kommt man in den Löschmodus.

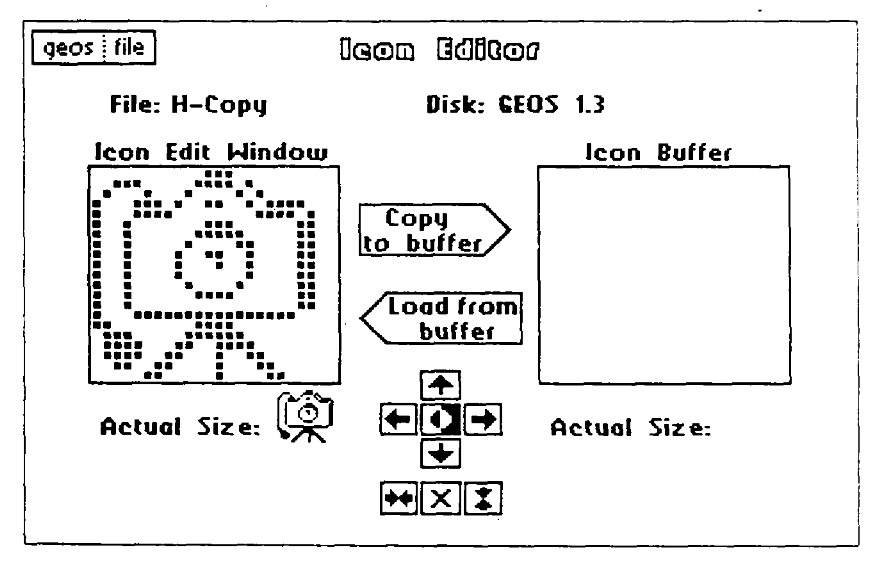


Bild 1.8.4.5: Arbeiten mit dem ICON EDITOR

Wir malen jetzt einmal ein Bild, das wie eine Kamera aussieht und uns als ICON für ein Hardcopy-Programm dienen kann. Vergleichen Sie hierzu die entsprechende Abbildung im Buch. Das fertige Bild kann man nun durch Anklicken des entsprechenden Pfeiles zwischen den Arbeitsfenstern in den ICON BUFFER verlegen, um es im gegenwärtigen Stand dort zu sichern und links noch weitere Veränderungen auszuprobieren. Sollten diese uns nicht gefallen, können wir durch Anklicken des Gegenpfeils den alten Zustand wiederherstellen.

Bleiben wir also bei unserem Kamera-ICON. Durch Anklicken des Feldes "Save Icon" unter FILE kann man es nun abspeichern. Dadurch hat dann das vorher ausgewählte File das gerade erstellte ICON bekommen. Durch "Recover ICON" können Sie eine schon früher abgespeicherte Version wieder hervorholen.

Und durch das Anklicken des "Remove ICON"-Feldes erscheint ein Dialogfenster mit der Frage: "Are you sure you want to delete this file's header?". Its Icon and file attribute information will be lost." Das bedeutet:

Wenn man also z.B. ein File in die ursprüngliche Nicht-GEOS-Version zurückverwandeln, d.h. auch den Info-Sektor löschen will, erreicht man das durch Anklicken von "YES". Anschließend hat man dann wieder ein ganz normales C64-File. Das heißt, man ist dann wieder im Ausgangszustand, bevor man begonnen hat, mit dem ICON EDITOR zu arbeiten. Der letzte Punkt unter FILE heißt "QUIT", und mit dem kann man dann den ICON EDITOR auch wieder verlassen. Man kehrt dann ins DESKTOP zurück.

GRAPHICS GRABBER

Mit diesem Programm haben Sie die Möglichkeit, Grafiken, die Sie mit anderen Malprogrammen als GEOPAINT erstellt haben, an GEOS anzupassen. Anschließend können Sie sie dann etwa mit GEOPAINT weiterbearbeiten.

Was den Namen angeht, so kommt er von dem englischen Wort "to grab", was soviel bedeutet wie "an sich reißen" oder "schnap-

pen". Das klingt zwar etwas dramatisch, ist aber eine grundsolide Angelegenheit. In der Version, die wir hier im DESK PACK 1 vorfinden, kann der GRAPHICS GRABBER Grafiken der folgenden Mal- und Zeichenprogramme konvertieren:

- PRINTMASTER
- PRINTSHOP und
- NEWSROOM.

Nach der Installation (vgl. dazu die Beschreibung bei der Vorstellung von GEOS V1.3!) und dem Laden durch Doppelklicken erscheint ein leeres Bild. Oben hat man eine Menüleiste mit den Punkten GEOS, FILE, SAVE und OPTIONS.

Unter GEOS wird Ihnen zunächst ein GRABBER-Info angeboten. Anschließend sind die übrigen Accessories aufgelistet, die noch da sind. In diesem Falle handelt es sich zum Beispiel um das schon vorgestellte Kalenderprogramm sowie um das BLACK-JACK-Spielprogramm. Nachdem wir nun FILE angeklickt haben, haben wir folgende Felder zur Auswahl:

- PRINTMASTER,
- PRINTSHOP und
- NEWSROOM.
- Ganz unten haben wir noch das bekannte QUIT-Feld.

Wir wählen z.B. PRINTSHOP und werden anschließend aufgefordert, eine Diskette mit PRINTSHOP-Grafiken einzulegen (Insert a disk containing a printshop graphik in drive A!). Anschließend sehen wir in einer Box die auf der Diskette vorhandenen entsprechenden Grafiken und können eine auswählen. Sollte die von uns gewünschte unter den fünf angezeigten Files nicht sein, können wir durch Betätigen der Scroll-Pfeile auch die übrigen sehen.

Sollten Sie entgegen unseren Voraussagen keinen einzigen Eintrag in der File-Select-Box sehen, so befindet sich auch keine PRINTSHOP-Grafik auf dieser Diskette. Und nur die werden hier angezeigt. Als Auswahlmöglichkeit haben wir jetzt:

GRAB und CANCEL.

Wir klicken natürlich GRAB an, weil wir ja eine Grafik umwandeln wollen. Links neben der nun auftauchenden ausgewählten Grafik erscheint nun eine neue Menüleiste mit folgenden Punkten (in der Reihenfolge von oben nach unten):

1. Schließsymbol: Mit ihm kommen Sie wieder zur File-Select-Box, d.h., eine Ebene höher, können dann also neue Grafiken auswählen.
2. Schere: Hier können Sie aus der Grafik etwas ins PHOTO SCRAP ausschneiden.
   Dasselbe erreichen Sie auch mit Hilfe des Menüpunktes SAVE, wenn Sie dort "in a scrap" anklicken.
3. Album: Hier können Sie etwas aus der Grafik direkt in ein vorher ausgewähltes PHOTO ALBUM einkleben.
   Dasselbe erreichen Sie auch mit Hilfe des Menüpunktes SAVE, wenn Sie dort "in an album" anklicken.
4. Hier sehen Sie eine Art aufgeschlagenen Kalender mit den Begriffen "first", "prev", "next" und "last" und können entsprechend zur ersten, zur vorausgehenden, zur nächsten und zur letzten der (in diesem Falle) PRINTSHOP-Grafiken wechseln.

Was das Obermenü "SAVE" angeht, so können wir einmal die konvertierte Grafik "in a scrap", d.h. also ins PHOTO SCRAP, bzw. "in an album", d.h. in ein Fotoalbum legen.

Diesen Menüpunkten entsprechen in der Menüleiste links zwei Anwählfelder (s.o.). Unter OPTIONS können Sie einmal ein bereits existierendes Album wählen, CHOOSE AN ALBUM, oder ein neues Album anlegen, CREATE A NEW ALBUM.

BLACK JACK

Stellen Sie sich vor, Sie haben an einem Tag hunderte von PRINTSHOP-Bildern mit Hilfe des GRAPHICS GRABBERS konvertiert. Dann haben Sie sicherlich Entspannung nötig. Und genau dafür ist das BLACK JACK DESK ACCESSORY gedacht. So, wie PC-Besitzer z.B. ihren "Flight Simulator" haben, können

Sie nun unter GEOS gegen den Computer Karten spielen und (hoffentlich) gewinnen. Nach der Installation (vgl. dazu die Beschreibung bei der Vorstellung von GEOS V1.3!) und dem Laden durch Doppelklicken erscheint zunächst ein Fenster als Grundbild. Oben links haben Sie eine Menüzeile mit den Punkten FILE und PLAY.

Wenn man FILE anklickt, hat man die Möglichkeit, das Spiel über QUIT (im Gegensatz zu dem realen Black-Jack-Spiel "ungeschoren") zu verlassen. Unter PLAY verbirgt sich nur der Programmpunkt START.

Die Fläche des Fensters wird wieder durch die uns schon von anderen GEOS-Erweiterungen her bekannte waagrechte Dreifachlinie in zwei Teile geteilt.

Klickt man unter PLAY das Feld START an, werden unsichtbar, aber gut hörbar die Karten gemischt und abgehoben. Daß der Rechner zugange ist, erfährt man als Spieler nicht nur durch charakteristische Geräusche, sondern auch durch kurz unter der Dreifachlinie eingeblendete Mitteilungen ("SHUFFELING" für Mischen und "CUTTING" für Abheben).

(Was das korrekte Mischen und Abheben angeht, müssen wir allerdings Berkeley Softworks vertrauen.) Unter der Dreifachlinie finden sich links dann anschließend zwei Anzeigen: zum einen BET, was den Einsatz darstellt (am Anfang: 10), zum anderen ACCT für das noch vorhandene Vermögen des Spielers (am Anfang: 1000). Rechts davon erscheint noch einmal BET, hier jetzt zunächst ohne Betrag. Ganz rechts gibt es noch zwei Anklickfelder: ALTER und SAME. Klickt man SAME an, bedeutet das, daß Sie als Spieler mit dem Einsatz einverstanden sind. Klickt man dagegen ALTER an, kann anschließend per Tastatur ein Teil des Vermögens nach freier Wahl eingesetzt werden.

In unserem ersten Probedurchgang klicken wir erst einmal SAME an. Begleitet von anheimelnden Tönen fliegen dann von rechts quer über den Bildschirm je zwei Karten heran und ordnen sich links auf dem Bildschirm zu zwei Paaren an.

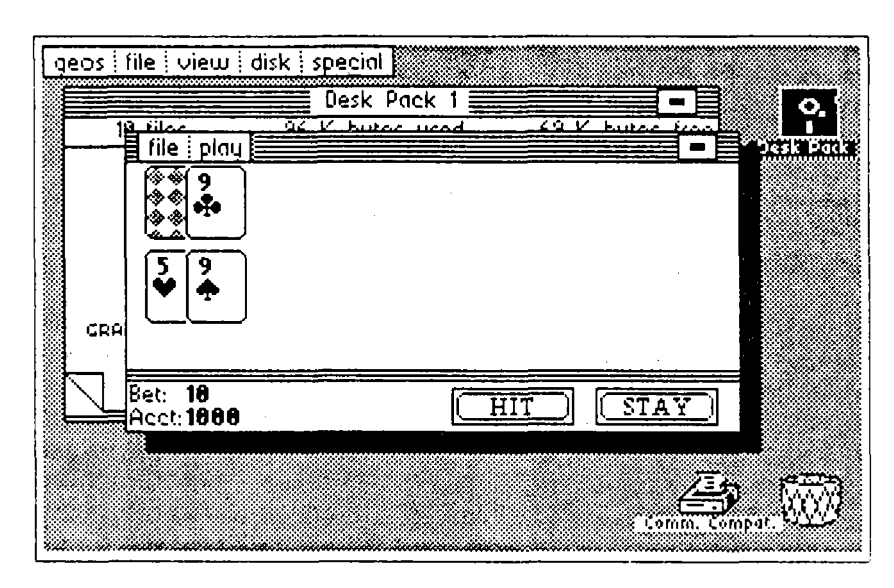


Bild 1.8.4.6: BLACK JACK - Spielbeginn

Das obere Paar, von dem die linke Karte verdeckt ist, gehört dabei dem Gegenspieler, d.h. dem Computer. Die beiden unteren sollte man so sorgfältig betrachten, daß man entscheiden kann, ob man jetzt rechts HIT anklickt oder STAY.

Im ersten Fall möchte man eine weitere Karte (HIT), im zweiten Fall (STAY) ist man mit seinen bisherigen Karten zufrieden und der Computer beginnt zu ziehen. Ist auch der Computer zufrieden, wird der Gewinner durch ein

* Winner *

angezeigt. Durch Anklicken von DEAL schiebt der Computer dann auch die zugehörigen Beträge auf das entsprechende (hoffentlich unser) Konto. Neben den Grundmöglichkeiten gibt es einige Besonderheiten, die dem Spiel besonderen Pfiff geben: So kann man z.B. seine Karten auf zwei Hände verteilen (SPLITTING THE HAND). Das ist möglich, sobald man zwei gleiche Karten hat, mit ihnen kann einzeln weitergespielt werden.

In unserem Falle haben wir das bei zwei Assen natürlich sofort gemacht, prompt bei beiden 21 gemacht und so doppelt gegen die Computerbank gewonnen. Dieser seltene Fall war uns natürlich sofort eine Abbildung wert. Wann hat man sonst schon mal die Möglichkeit, Glück im Spiel so zu verewigen?

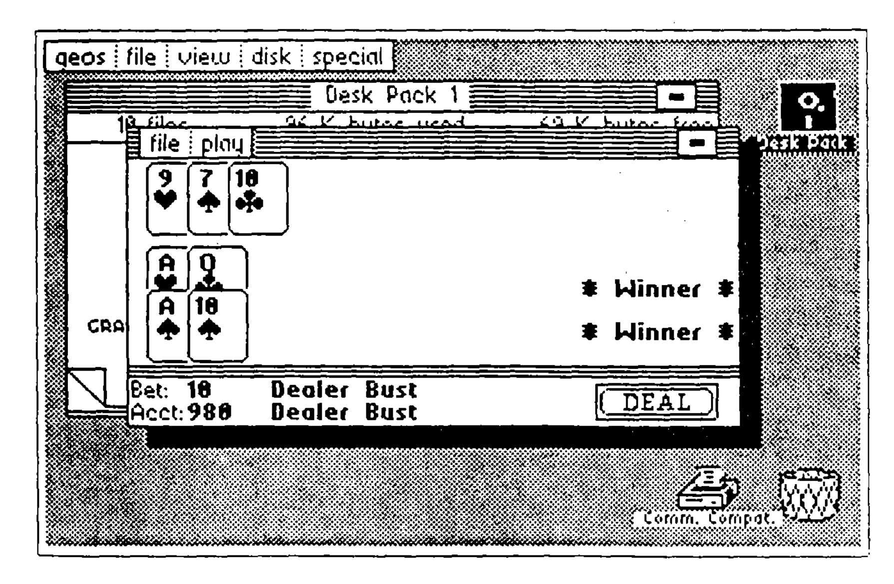


Bild 1.8.4.7: BLACK JACK - Splitting The Hand

Als zweite Möglichkeit gibt es noch DOUBLE DOWN. Wenn man mit den ersten beiden Karten 10 oder 11 Punkte hat, wird man gefragt, ob man DOUBLE DOWN möchte.

Obwohl DOUBLE DOWN ja eigentlich "Doppelt abwärts" heißt, wird in diesem Falle die Wette verdoppelt und automatisch eine einzige weitere Karte gezogen. Dann ist der Gegenspieler dran. Als letzte besondere Möglichkeit kann INSURANCE auftauchen. Wenn die gezeigte Karte der Bank ein "As" ist, kann man diese Möglichkeit anwählen. Das kostet zwar die Hälfte des Wetteinsatzes. Schafft die Bank aber einen BLACK JACK, so ist man dagegen versichert (INSURANCE) und verliert nichts außer der Versicherungssumme. Macht die Bank keinen BLACK JACK, dann verliert man das INSURANCE, aber das Spiel geht ganz normal weiter, d.h. man kann die Wette auch gewinnen.

Insgesamt hat man mit diesem Accessory nun auch unter GEOS den Einstieg in die Spielwelt gefunden, getreu wohl dem Grundsatz: Wer hart arbeitet, soll auch seinen Spaß haben. Besonders gut gefallen hat uns bei diesem Spiel das grafische und akustische Drumherum, das dem Spiel einen zusätzlichen Reiz verleiht und zu einer netten Form der Entspannung macht, vor allem wird man unter GEOS natürlich auch vor den unter Umständen problematischen Folgen eines Spielbankbesuchs bewahrt. Man kann nur die 1000 Dollar verlieren, die man sich vorher unter START geholt hat.

1.8.5 GEOPUBLISH

GEOPUBLISH ist ein Desktop-Publishing-Programm für die GEOS-Serie. Desktop Publishing, was übersetzt so viel wie Veröffentlichungen vom Schreibtisch aus heißt, ist eine Programmart, mit der man Zeitungen, Bücher und Plakate - kurz gesagt, alles, bei dem man layouten muß - erstellen kann. Sie können Texte und Grafiken zusammen in einem Dokument verarbeiten, beliebig miteinander in Verbindung bringen, ja sogar übereinanderlegen. Diese Art der Anwendung war bis jetzt nur auf größeren Rechnern mit mehr "Rechenkraft" möglich, doch GEOPUBLISH zeigt, daß es auch mit dem C64 und GEOS gut möglich ist. Da GEOPUBLISH sehr umfangreich ist, würde eine genaue Beschreibung der Funktionen den Rahmen dieses Kapitels sprengen. Deshalb erläutere ich hier nur die wichtigsten Funktionen.

Installation von GEOPUBLISH

Die Installation von GEOPUBLISH ist recht einfach: Starten Sie GEOS ganz normal, nehmen Sie die Systemdiskette aus dem Laufwerk und legen Sie die Orginaldiskette von GEOPUBLISH ohne Schreibschutz ins Laufwerk und "ÖFFNEN" sie. Starten Sie nun GEOPUBLISH durch einen Doppelklick. Das Laufwerk läuft nun an, und nach einiger Zeit erscheint die Meldung, daß GEOPUBLISH installiert wurde. Das Programm wird nun aber abgebrochen. Nachdem Sie wieder auf dem Desktop angelangt sind, können Sie GEOPUBLISH ganz normal starten.

Arbeiten mit GEOPUBLISH

Laden Sie nun GEOPUBLISH durch einen Doppelklick auf dessen Icon. Nach kurzer Ladezeit meldet sich das schon von GEOPAINT und GEOWRITE bekannte Fenster, in dem Sie entscheiden müssen, ob Sie ein neues Dokument erstellen wollen (CREATE), ein schon existierendes öffnen wollen (ÖFFNEN) oder GEOPUBLISH verlassen wollen. Klicken Sie nun bitte CREATE an, und tragen Sie einen Namen für Ihr Dokument ein. Daraufhin meldet sich GEOPUBLISH mit folgendem Bild:

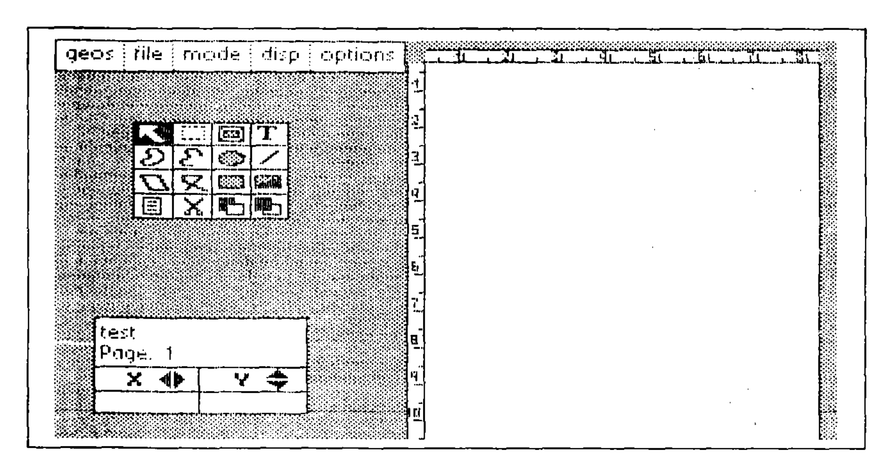


Bild 1.8.5.1: GEOPUBLISH-Grundbildschirm

Oben sehen Sie die für GEOS-Programme schon traditionelle Menüzeile, die folgende Punkte enthält: GEOS, FILE, MODE, DISK, OPTIONS. Hier sei gleich schon angemerkt, daß es nicht immer möglich ist, alle Punkte anzuwählen. Ist ein Punkt nicht anwählbar, so wird der Menüpunkt "kursiv" geschrieben. Auf die einzelnen Punkte werde ich später noch einmal eingehen. Darunter sehen Sie einen Werkzeugkasten, der dem aus GEOPAINT in machen Dingen ähnlich ist. Darunter ist ein Kästchen, in dem der Name des Dokumentes angegeben ist. Darunter befindet sich die Seitennummer und ein Koordinatenfeld, in dem bis jetzt noch nichts steht außer einem einsamen X und einem Y. Die ganze rechte Hälfte wird von einer Übersicht über Ihr Dokument belegt, welches jetzt noch logischerweise leer ist.

Fahren Sie nun einmal mit der Maus über das Blatt, und Sie werden sehen, daß sich die Zahlen im Koordinatenfeld ändern.

Wir sollten zuerst einmal das Layout des Dokumentes festlegen. Klicken Sie dazu bitte den Punkt PAGE LAYOUT in dem Menü MODE an. Es erscheint wieder eine leere Seite und ein anderer Werkzeugkasten. Sie können schon fertige Layouts laden, indem Sie im Menü FILE den Punkt LIBRARY anklicken. Es erscheint ein Fenster, in dem Sie sich ein Layout aussuchen können. Sie haben die Wahl zwischen ein- bis vierspaltigen Layouts. Am besten wird es sein, einfach einmal die verschiedenen auszuprobieren, indem Sie den Namen und dann das Feld ÖFFNEN anklicken. Es erscheint noch eine Warnung, die besagt, daß das schon bestehende Layout gelöscht wird. Klicken Sie bitte das Feld OK an. Die Floppy läuft nun kurz an, und es wird das Layout auch auf der Seite dargestellt

Um nun wieder zu unserem alten Bildschirm zurückzukommen, aktivieren Sie bitte PAGE GRAPHICS im Menü MODE. Sie sehen dann das noch leere Dokument, auf dem die Spalten angezeigt werden.

Da wir aber nicht immer ein leeres Blatt haben wollen, werden wir ein paar Kreise und Striche ziehen. Klicken Sie bitte das Kreissymbol im Werkzeugkasten an und fahren Sie mit der Maus auf das Blatt. Der Mauszeiger ändert sich in ein Fadenkreuz. Wenn Sie die Maustaste drücken, so läuft die Floppy kurz an und Sie können - genauso wie in GEOPAINT - einen Kreis zeichnen. Ist der Kreis nach Ihren Wünschen, so drücken Sie die linke Maustaste, und er wird entgültig gezeichnet. Die Linien- und Viereckfunktionen funktionieren genauso wie in GEOPAINT, weshalb ich sie hier nicht noch einmal extra erkläre. Das Freihandzeichnen ist im Gegensatz zu GEOPAINT etwas anders. Klicken Sie bitte das Symbol dafür an (es ist das 2. von links in der zweiten Reihe), fahren Sie mit der Maus auf das Blatt, und drücken Sie den Feuerknopf. Die Floppy läuft wieder kurz an. Sie können nun eine Linie ziehen. Soll dort Ihr erster Wendepunkt sein (warum ich diesen Punkt so nenne, werden Sie gleich selber sehen), so drücken Sie bitte den Knopf. Die Linie verschwindet daraufhin. Fahren Sie mit der Maus zum zweiten

Wendepunkt. Wie Sie sehen, wird die Ecke nun abgerundet, so daß der Eindruck einer von Hand gezeichneten Linie entsteht. Sie können bei dieser Funktion beliebig viele Wendepunkte bestimmen. Soll die Linie die endgültige Form annehmen, drücken Sie bitte den Feuerknopf am Endpunkt zweimal kurz hintereinander. Die Linie wird nun entgültig gezeichnet. Nach dem gleichen Prinzip arbeitet auch die Funktion links neben der Freihandfunktion mit dem Unterschied, daß Anfangs- und Endpunkt miteinander verbunden werden. Ähnlich sind die Funktionen unter den Freihandfunktionen, nur daß hier die Ecken nicht abgerundet werden.

Ich werde nun etwas genauer auf die Funktionen eingehen, die ein DTP-Programm (DTP ist die Abkürzung für Desktop Publishing) charakterisieren. Das wichigste ist, daß ein gemaltes Objekt (z.B. eine Linie) nicht zur Gesamtzeichnung gehört, sondern ein eigenes Objekt für sich ist. Nur dadurch ist es möglich, einzelne Zeichnungen oder Textpassagen zu verschieben. Klicken Sie nun bitte das Pfeilsymbol oben links im Werkzeugkasten an. Fahren Sie dann bitte mit der Maus auf einen gerade gezeichneten Kreis, und klicken Sie ihn einmal an. Sofort bildet sich ein Viereck um ihn, in dessen oberen linken und unteren rechten Ecke je ein kleines Quadrat ist. Klicken Sie bitte zuerst das obere an. Die Floppy läuft kurz an, und daraufhin verschwindet der Kreis. Bewegen Sie nun die Maus, so bewegt sich auch das Viereck. Haben Sie das Quadrat an die neue Stelle gebracht, so drücken Sie noch einmal den Knopf. Die Floppy läuft kurz an, und der Kreis wird an der neuen Stelle gezeichnet. So lassen sich leicht alle Objekte verschieben.

Mit dem unteren kleinen Quadrat läßt sich ein Objekt in alle möglichen Richtungen dehnen. Klicken Sie dazu das untere Quadrat an, und bewegen Sie die Maus. Es erscheint daraufhin wieder ein Viereck, welches sich nun durch Bewegen der Maus beliebig vergrößert oder verkleinert. Wenn Sie wieder den Knopf drücken, so wird das Objekt - der Größe des Vierecks entsprechend - gezeichnet.

Ihr Bildschirm sieht nun bestimmt konfus aus, denn hat man ein Objekt bewegt, so werden die Teile von anderen Objekten, die

im gleichen Bereich lagen gelöscht, nach dem Verschieben jedoch nicht wieder hergestellt. Um das zu tun, gibt es eine Funktion, die das Blatt wieder richtig herstellt. Klicken Sie es einfach an (es ist das Symbol unten rechts), und nach kuzem Laufen der Floppy ist das Blatt wieder hergestellt.

Objekte lassen sich auch sehr einfach löschen. Klicken Sie dazu das auszuschneidende Objekt und dann das Symbol an, welches eine Schere darstellt, und das Objekt verschwindet sofort.

Eine Möglichkeit, Grafiken einzubinden, besteht darin, die einzubindende Grafik ins PHOTO SCRAP abzulegen (das muß natürlich schon vor dieser Sitzung geschehen sein) und dann das dritte Symbol von links in der ersten Reihe anzuklicken. Fahren Sie nun mit der Maus auf die Stelle, an der die Grafik eingebunden werden soll, und drücken Sie den Feuerknopf. Die Grafik wird nun in Ihr Dokument eingebunden. Gab es jedoch kein PHOTO SCRAP, so erscheint folgende Fehlermeldung:

```
NO PHOTO SCRAP
```

Wollen Sie mehrere Objekte auf einmal bearbeiten, können Sie mit dem Markierer gleich mehrere Objekte auswählen. Klicken Sie einfach dessen Symbol an (1. Reihe; 2. von links), und markieren Sie die Objekte auf die gleiche Art, wie Sie in GEOPAINT auch Bildbereiche markieren. Nun können Sie diese Objekte alle auf einmal bearbeiten.

Texte in GEOPUBLISH

Bisher haben Sie nur Bilder malen bzw. verarbeiten können, doch sollten Zeitungen oder Plakate auch Texte enthalten. Kein Problem, denn wir haben ja GEOPUBLISH. Sie haben zwei Möglichkeiten, Text in Ihr Dokument zu bekommen. Klicken Sie im Werkzeugkasten bitte auf das Symbol, welches ein T darstellt. Fahren Sie nun mit der Maus auf die Stelle am Bildschirm, an der der Text stehen soll. Drücken Sie wieder den Feuerknopf. Daraufhin erscheint ein Fenster, in dem ein Cursor steht. Unten, am Rand des Fensters, gibt es drei Felder zum Anklicken: OK, ATTR, ABBRUCH. Wie unschwer zu erraten, ist das OK-Feld

dazu da, Ihre Eingaben zu bestätigen, während das ABBRUCH-Feld die ganze Sache abbläst. Klicken Sie auf das ATTR-Feld so erscheint folgendes Fenster:

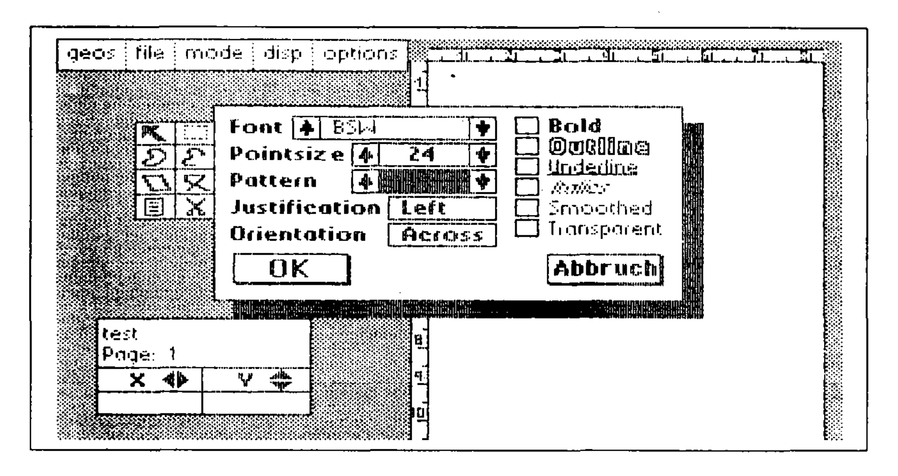


Bild 1.8.5.2: Textattribut-Fenster

Hier können Sie nun eingeben, welche Schriftart (Font) Sie benutzen wollen, wie groß Ihre Schrift (Pointsize) ist, aus welchem Muster (Pattern) Ihre Schrift sein soll (ja, Sie haben richtig gehört), in welche Richtung geschrieben werden soll (Orientation), wie der Text justiert werden kann (Justification) und in welchem Stil geschrieben werden soll. Um die verschiedenen Punkte einzustellen, gibt es Scroll-Pfeile oder einfach Schalter. Die Scrollpfeile funktionieren genauso wie beim Dokumentauswahl-Fenster in GEOPAINT. Die Schalter sind einfach anzuklicken, und schon ändert sich diese Einstellung entsprechend. Andere Schalter werden beim Anklicken mit einem Kreuz versehen, welches anzeigt, daß diese Option nun aktiviert ist. Noch ein Wort zu "Pointsize" und "Pattern", da diese Optionen wirklich neu sind. Mit Pointsize haben Sie die Möglichkeit, die Größe der Schrift beliebig zu ändern. Mit Pattern können Sie Ihre Schrift durch Muster ersetzen lassen. Haben Sie zum Beispiel als Muster Rauten ausgewählt, so wird Ihre Schrift aus lauter kleinen Rauten zusammengesetzt.

Um nun auch einen Text schreiben zu können, müssen Sie wieder ins Editierfenster zurück. Klicken Sie dazu das Feld EDIT an, und Sie sind wieder im altbekannten Editierfenster. Sie können hier Ihre Texte schreiben und editieren. Klicken Sie im Attributfenster jedoch nicht EDIT, sondern OK an, so verlassen Sie das Attributfenster und befinden sich wieder auf der Arbeitsfläche. Ihre Änderungen bleiben jedoch erhalten. Wählen Sie ABBRUCH, so werden alle Änderungen ignoriert, und Sie befinden sich wieder auf der Arbeitsfläche. Diese Attributänderungen können Sie übrigens auch bei anderen Werkzeugen machen. Klicken Sie dazu einfach das entsprechende Symbol im Werkzeugkasten an, und klicken Sie dann auf das rechte Feld in der dritten Reihe. Nun können Sie für dieses Werkzeug verschiedene Änderungen vornehmen.

Die zweite Art, Texte in Ihr Dokument einzubinden, besteht darin, schon fertige Texte, die z.B. mit GEOWRITE erstellt wurden, zu nutzen. Um dieses zu tun, klicken Sie bitte den Menüpunkt PAGE LAYOUT in dem Menü MODE an. Es erscheint wieder der Werkzeugkasten für die Layouterstellung. Klicken Sie bitte das Symbol an, welches ein T darstellt. Die Floppy lädt nun die Namen der Text-Files, die sich auf der Diskette befinden, und zeigt sie an. Sie können sich wie bei GEOWRITE einen Text aussuchen und laden. Das Fenster verschwindet wieder, doch wo ist der Text? Kein Problem! Klicken Sie einfach in der Spalte, in der der Text erscheinen soll. Sie wird dann mit einem Muster gefüllt, was heißt, das in dieser Spalte nun Text steht.

Kehren Sie bitte zu unserem Arbeitsblatt zurück, in dem Sie im Menü MODE den Punkt PAGE GRAPHICS anklicken.

Sonderfunktionen von GEOPUBLISH

Hier werde ich nun noch einige Sonderfunktionen von GEOPUBLISH erläutern. An dieser Stelle sei noch einmal erwähnt, daß nicht alle Features angesprochen werden können, da das den Rahmen dieses Buches sprengen würde.

Im Menü FILE wurde schon ein neuer Menüpunkt erklärt. Ein zweiter neuer ist der Punkt DOC SETUP. Klicken Sie ihn an, so können Sie verschiedene Voreinstellungen für Ihr Dokument einstellen. Die restlichen Menüpunkte sind die gleichen wie unter GEOWRITE bzw. GEOPAINT.

Im Menü MODE können Sie noch eine MASTER PAGE erstellen, mit der Sie, wenn Sie mehrere Seiten erstellen wollen, ein Grundlayout festlegen können, welches dann für jede Seite gilt.

Im Menü DISP können Sie beeinflussen, was während Ihrer Arbeit angezeigt werden soll. Außerdem haben Sie die Möglichkeit, Ihr Dokument zu vergrößern, um Feinarbeiten auszuführen. Dies geschieht mit dem Menüpunkt ZOOM. Die Übersicht über Ihr Dokument bekommen Sie wieder durch PREVIEW. Mit den anderen Funktionen können Sie einstellen, ob z.B. die Grafiken während Ihrer Arbeit angezeigt werden sollen oder ob an deren Stelle nur ein Quadrat steht, was jedoch schneller ist. Diese Menüpunkte haben auch eine Schalterfunktion. Ist ein Punkt aktiv, so steht ein Sternchen vor dem entsprechenden Punkt. Ist er inaktiv, so steht - man höre und staune - kein Sternchen vor diesem Punkt. Im Menü OPTIONS haben Sie noch die Möglichkeit, direkt zu bestimmten Seiten zu springen (existierte die Seite vorher noch nicht, so werden Sie gefragt, ob sie erstellt werden soll), den Werkzeugkasten ein- und auszublenden (TOOLBOX), die Koordinatenanzeige auszustellen (SNAP), ein Gitter zu erstellen (SET RATCHET), in dem sich die Maus nur bewegen kann (wodurch symmetrisches Arbeiten vereinfacht wird) und die Abstände von den Rändern zur Spalte festzusetzen (SET GUTTERS) und den Rahmen auszublenden (RULERS).

Fazit

Alles in allem halten wir GEOPUBLISH für eine lohnende Anschaffung, da es nun mit dem C64 möglich ist, Texte professionell zu gestalten, was bisher nur auf größeren (und teureren) Systemen möglich war. Mit GEOPUBLISH stößt man in einen ganz neuen Bereich der Textbe- und -verarbeitung, was sich in jedem Falle für den Anwender lohnen wird.

1.8.6 GEOCALC

GEOCALC ist ein weiteres Produkt aus der GEOS-Reihe. Bei GEOCALC handelt es sich um eine Tabellenkalkulation. "Was ist eigentlich eine Tabellenkalkulation?" werden sich nun vielleicht manche Leser fragen. Nun, teilt man das Wort "Tabellenkalkulation", so erhält man das Wort Tabelle und Kalkulation. Was eine Tabelle ist, brauche ich wohl nicht weiter zu erklären. Unter einer Tabellenkalkulation versteht man also eine Tabelle, mit der man kalkulieren oder - anders ausgedrückt - rechnen kann. Sie können also einzelne Rechenfelder miteinander logisch verknüpfen. Als Beispiel sei hier ein Kassenbon angeführt:

```
 10.34 DM
+20.49 DM
---------
 30.73 DM
 40.00 DM
---------
- 9.27 DM
```

In den ersten beiden Zeilen sind hier die Preise für zwei Artikel angegeben. Die dritte Zeile gibt daraufhin die Summe aus, die sich aus der Addition von Zeile 1 und Zeile 2 ergab. In der vierten Zeile wird eingetragen, welche Summe man der Kassiererin gegeben hat. In der fünften Zeile wird die Geldsumme angegeben, die der Kunde nun wiederbekommt. Diese Geldsumme errechnet sich aus folgender Formel: 3. Zeile - 4. Zeile. Genauso wird auch bei einem Kalkulationsprogramm vorgegangen, was wir uns nun genauer anschauen werden.

Installation von GEOCALC

Die Installation von GEOCALC verläuft genauso wie die von GEOPUBLISH. Starten Sie also GEOCALC mit einem Doppelklick. Nach einiger Zeit erscheint dann die Meldung, daß GEOCALC installiert ist. Daraufhin kehrt GEOS wieder zum Desktop zurück. Von nun an können Sie GEOCALC ganz normal starten.

Funktionen von GEOCALC

Haben Sie GEOCALC durch einen Doppelklick gestartet, so erscheint das bekannte Fenster, in dem Sie entscheiden müssen, ob Sie ein neues Dokument erstellen wollen (CREATE), ob Sie ein schon bestehendes Öffnen wollen (ÖFFNEN) oder ob Sie zum Desktop zurückkehren wollen. Klicken Sie das Feld CREATE an, und geben Sie dem Dokument den Namen Kassenzettel, denn wir werden nun versuchen, eine solches Beispiel in die Tat umzusetzen. Es erscheint folgender Bildschirm:

Bild 1.8.5.3: Grundbildschirm GEOCALC

Oben am Bildschirmrand sind wieder fünf Menüs angegeben: GEOS, FILE, EDIT, OPTIONS, DISPLAY. Darunter ist eine Zeile zu sehen, in der A1, zwei Symbole und der Cursor stehen. Wir werden diese Zeile im folgenden "Eingabezeile" nennen. Darunter sehen Sie ein Koordinatenfeld, welches wir einfach als "Arbeitsblatt" bezeichnen werden. Ganz unten am Bildschirmrand sind noch einzelne Symbole zu erkennen.

Das Prinzip einer Tabellenkalkulation ist folgendermaßen: Das Arbeitsblatt ist eingeteilt in einzelne Felder, die wie beim Schachbrett durch eine Zahl, welche für die Zeile steht, und einen Buchstaben, welcher für die Spalte steht, benannt sind. In diese Felder kann man nun Formeln, wodurch sich dann der

"Feldinhalt" ergibt, Texte oder einfach Zahlen eingeben. Diese Felder lassen sich nun beliebig miteinander verknüpfen.

Die Zeichenkombination A1 gibt an, welches Feld gerade aktuell ist. Dieses Feld ist auch auf dem Arbeitsplatz schwarz umrandet. Dies ist nach dem Start immer das Feld A1. Um nun ein anderes Feld zu aktivieren, müssen Sie nur mit der Maus auf das gewünschte Feld fahren und es anklicken. Daraufhin wird dieses schwarz umrandet und dessen Koordinaten stehen nun in der Eingabezeile.

Da wir auch später noch wissen wollen, um was für ein Dokument es sich hier handelt, sollten wir ihm erst einmal die Überschrift "Kassenbon" geben. Aktivieren Sie bitte Feld A1, und tippen Sie das Wort "Kassenbon" ein. Ihre Eingaben erscheinen zuerst in der Eingabezeile. Haben Sie sich vertippt, so können Sie Zeichen durch die <Backspace>-Taste löschen. Wollen Sie nur einen Buchstaben einfügen, so klicken Sie einfach die Stelle an und schreiben Sie den neuen Text. Um Ihre Eingaben zu bestätigen, müssen Sie die <Return>-Taste drücken. Das Wort Kassenbon wird im Feld A1 angezeigt, und Feld A2 wird nun aktiviert. In diesem Feld sollte nichts stehen, da sich die Überschrift ja von dem restlichen Kassenbon abheben soll. Drücken Sie deshalb einfach nur <Return>. In das dritte, vierte und fünfte Feld können Sie nun "Ihre" Preise für Kaffee, Milch und Kuchen eintragen. Tippen Sie jeweils einfach nur die jeweilige Zahl ein. Im sechsten Feld tragen wir einen Strich ein, da sich die Summe ja von den restlichen Feldern abheben soll. Mittlerweile sind wir im Feld A7 angelangt. Nun wird's erst richtig interessant, denn hier soll der zu bezahlende Betrag stehen, sprich: die Summe von Feld A3, A4 und A5. Tippen Sie nun folgendes ein:

```
=A3+A4+A5
```

Diese Zeile bedeutet, das in diesem Feld die Summe von A3, A4 und A5 stehen soll. Daß es sich hierbei um eine Formel und nicht um einen einfachen Text handelt, erkennt GEOCALC an dem vorangegangenen =-Zeichen. Übersetzen könnte man diese Formel mit: "Feld A7 ist gleich A3 plus A4 plus A5." Sie sehen,

es ist gar nicht so schwer, GEOCALC rechnen zu lassen. Sobald Sie <Return> drücken, steht im Feld A7 die Summe von A3+A4+A5.

Im Feld A8 sollten wir die Summe angeben, die der Kunde bezahlt hat. Im Feld A9 soll wieder eine Querlinie stehen, denn nun folgt das Restgeld, welches der Kunde wiederbekommt. Sind wir nun im Feld A10 angelangt, so werden wir unseren Kassenbon fertigstellen. Hier soll also das Rückgeld angegeben werden, welches der Kunde wiederbekommt. Es errechnet sich aus folgender Formel:

```
Gegebener Betrag - zu bezahlender Betrag
```

Überlegen Sie einmal, was wir in die Eingabezeile für dieses Feld schreiben könnten. Na, genau! Dort muß folgendes stehen:

```
=A8-A7
```

Haben Sie <Return> gedrückt, so erscheint der Restbetrag sofort im Feld A9. Dieses einfache Beispiel sollte Sie in die Tabellenkalkulation einführen. Im folgenden werden nur noch die Funktionen der einzelnen Menüpunkte angegeben. Hier noch ein Bild des fertigen Kassenbons:

geos | file | edit | options | display | Test

A1 Kassenbon

	A	B	C	D	E
1	Kassenbon				
2					
3	10.99				
4	9.89				
5	4.35				
6	----------				
7	25.23				
8	100				
9	----------				
10	74.77				
11					
12					
13					

Bild 1.8.5.4: Kassenbon

Funktionsübersicht

Fangen wir mit der Menüzeile an:

GEOS

Hier stehen die einzelnen Applikationen, die Sie auf der Diskette haben.

FILE

Hier stehen alles nur alte Bekannte, weshalb ich hier auch nichts erläutern muß. Eine Änderung gibt es bei dem Punkt PRINT. Hier erscheint nach dem Anklicken ein Menü, in dem Sie die Schriftqualität und noch einige andere druckerspezifischen Sachen einstellen können.

EDIT

Hier ist zu den alten Bekannten CUT, COPY und PASTE noch ein neuer Punkt hinzugekommen: Mit CLEAR können Felder schnell und einfach gelöscht werden. CUT, COPY und PASTE funktionieren genauso wie bei GEOPAINT und GEOWRITE, mit dem einzigen Unterschied, daß es sich hier um CALC SCRAPS handelt. Es werden - wenn vorhanden - die Formeln, nicht die sich daraus ergebenden Werte abgespeichert!

OPTIONS

Hier werde ich die einzelnen Punkte kurz erklären.

PASTE FUNCTION

Sie können Funktionen (z.B. SIN, COS etc.) in Ihre Formeln einfügen. Klicken Sie dazu diesen Punkt an, und wählen Sie sich die geeignete Funktion aus.

PASTE NAME

Die Felder können auch statt der Koordinatenbezeichnungen "Namen" erhalten. Sie können sich hiermit Namen, die Sie vorher eingegeben haben, aus einer Bibliothek aussuchen.

DEFINE NAME

Klicken Sie dieses Feld an, so eröffnet sich ein Fenster, in dem Sie einen Namen für das aktuelle Feld eingeben können. Dieser Name wird dann automatisch in die Bibliothek aufgenommen.

COPY TEXT SCRAP

Hiermit können Texte ins TEXT SCRAP zwischengespeichert werden.

PASTE TEXT SCRAP

Die Texte aus dem TEXT SCRAP können hiermit in das Dokument eingefügt werden.

NAMES OFF/ON

Die Namensgebungen werden mit diesem Menüpunkt aktiviert bzw. deaktiviert.

DISPLAY

Auch hier werde ich die einzelnen Punkte kurz erläutern.

FORMAT

Hiermit kann das Format bestimmt werden, in dem die Zahlen dargestellt werden sollen (z.B. als Prozentzahl mit x Nachkommastellen usw.).

STYLE

Hiermit können die Inhalte der Textfelder in Kursivschrift und/oder Fettschrift ausgegeben werden.

ALIGNMENT

Die Justierung des Textes der Felder kann bestimmt werden. Zur Auswahl stehen zentriert (CENTERED), linksbündig (LEFT JUSTIFIED) und rechtsbündig (RIGHT JUSTIFIED).

WIDTH

Hiermit kann die Breite der Felder eingestellt werden.

SCROLL ON/OFF

Bei ON erscheint links unten ein Fenster, welches das Blatt darstellt. Mit Hilfe eines kleinen Quadrates kann nun schnell an eine gewünschte Stelle gescrollt werden (siehe GEOPAINT). Bei OFF ist diese Funktion inaktiv.

In der ersten Zeile gibt es noch zwei Symbole, die ich noch kurz erläutern möchte: Beim Anklicken des "Kreuzes" wird eine Eingabe wieder rückgängig gemacht. Das Anklicken des "Hakens" bewirkt das gleiche wie die <Return>-Taste, jedoch bleibt das gerade behandelte Feld noch aktuell.

Wird das Feld angeklickt, welches das aktuelle Feld anzeigt, so erscheint an dessen Stelle ein Cursor. Gibt man nun eine Koordinate an, so wird direkt zu diesem Feld gesprungen.

Es können auch gleichzeitig mehrere Felder markiert werden, um bei allen z.B. die Schrift abzuändern. Klicken Sie dazu einfach das erste Feld an, halten Sie den Knopf gedrückt, und fahren Sie nun zum Endfeld. Die markierten Felder werden schwarz unterlegt. Durch Anklicken eines Buchstabens in der waagerechten Koordinatenreihe wird eine ganze Spalte markiert. Durch das Anklicken einer Zahl in der senkrechten Koordinatenreihe wird eine Zeile markiert. Klicken Sie das gestreifte Feld an, wird das ganze Dokument markiert. Durch Anklicken von CLEAR im EDIT-Menü kann so ein ganzes Dokument gelöscht werden. Klicken Sie das linke Symbol in der unteren Reihe an, so verwandelt sich der Pfeil in ein undefinierbares Objekt. Sie können ihn nur hoch und runter bewegen. Drücken Sie nun den Knopf, so wird Ihr Dokument ab dieser Zeile zwei-

geteilt. Sie können zwei Stellen Ihres Dokumentes im Auge behalten. Da diese Funktion so einzigartig ist, habe ich sie für Sie im Bild festgehalten:

geos | file | edit | options | display | Test

A1 Kassenbon

	A	B	C	D	E
1	Kassenbon				
2					
3	10. 99				
4	9. 89				
5	4. 35				
6	----------				
7	25. 23				

	A	B	C	D	E
1	Kassenbon				
2					
3	10. 99				
4	9. 89				
5	4. 35				

Bild 1.8.5.5: Splitting Screen

Rechts neben dem Symbol für den gespaltenen Bildschirm gibt es noch zwei weitere Symbole. Das erste, welches zwei Vierecke darstellt, ist ein Schalter, mit dem Sie die Darstellungsart von Feldern beeinflussen können. Hat ein Feld einen längeren Inhalt als das Feld selbst, so werden nämlich die restlichen Zeichen bei der Darstellung im Dokument abgeschnitten. Klickt man nun dieses Symbol an, so werden diese Zeichen nicht mehr abgeschnitten. Es werden also auch die "Überlängen" dargestellt.

Der Querstrich daneben ist einfach eine Anzeige, ob GEOCALC zur Zeit rechnet oder nicht. Da GEOCALC häufig viele Berechnungen durchführen muß, die dann auch ihre Zeit dauern, könnte man manchmal meinen, daß der Rechner abgestürzt ist. Dieses Symbol schafft in diesem Punkt Klarheit. Ist es invertiert, so wird gerade gerechnet, und der Rechner ist in dieser Zeit nicht mehr ansprechbar. Mit den vier Pfeilen rechts in der letzten Zeile kann der Ausschnitt gescrollt werden. Das Fahren der Maus an den Bildschirmrand, in dessen Richtung gescrollt werden soll, bewirkt das gleiche. Ich hoffe damit, allen Lesern einen Einblick in die Tabllenkalkulation gegeben zu haben. Vielleicht

ist damit dem ein- oder anderen klar geworden, daß ihm mit so einem Programm eine Menge Arbeit abgenommen werden kann. Insgesamt kann man sagen, daß dieses Programm eine gute Ergänzug zu der GEOS-Reihe darstellt, aber nicht unbedingt für jedermann nötig ist. Hat man jedoch eine Menge zu tabellieren, ist dieses Programm unerläßlich.

1.8.7 FONT PACK 1: Zwanzig Schriftarten unter GEOS

Sollten Sie sich nun mit BLACK JACK genügend entspannt haben und wieder daran denken, die Möglichkeiten von GEOWRITE zu nutzen, haben die Programmierer von Berkley Softworks Ihnen auch hier noch einiges anzubieten. Bisher konnten Sie bei GEOWRITE (und dementsprechend natürlich auch bei GEOPAINT) neben BSW noch fünf weitere Schriftarten verwenden. Mit dem neuen FONT PACK 1 werden Ihnen sage und schreibe gleich zwanzig neue Möglichkeiten zur Verfügung gestellt. Um Ihnen einen ersten Eindruck zu verschaffen, demonstrieren wir Ihnen auf den folgenden beiden Seiten ein paar von ihnen. Sie können dann selbst entscheiden, welche Schrift Ihnen am besten gefällt.

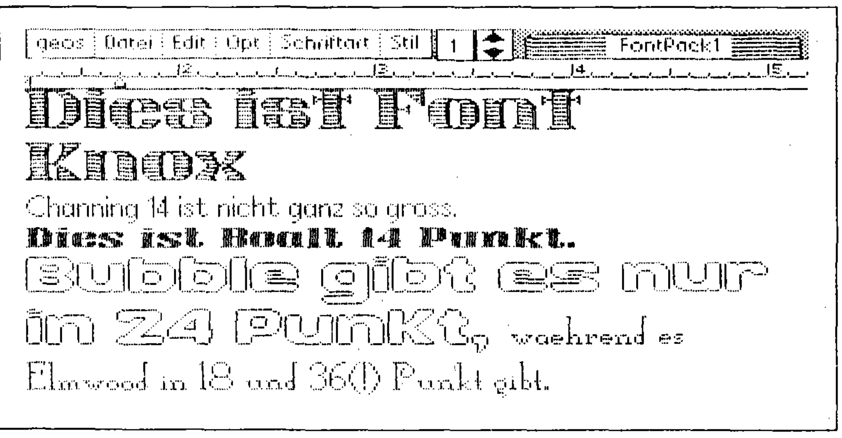


Bild 1.8.7.1: Neue Schriftarten: Font Knox, Channing, Boalt, Bubble, Elmwood

Font Knox ist sehr gut geeignet für Überschriften. Channing wirkt dagegen etwas gequetscht. Boalt ist genauso wuchtig wie Font Knox. Bubble erinnert stark an Kaugummis! Elmwood ist sehr "spinnenartig" und gibt es sogar in 36 Punkten!

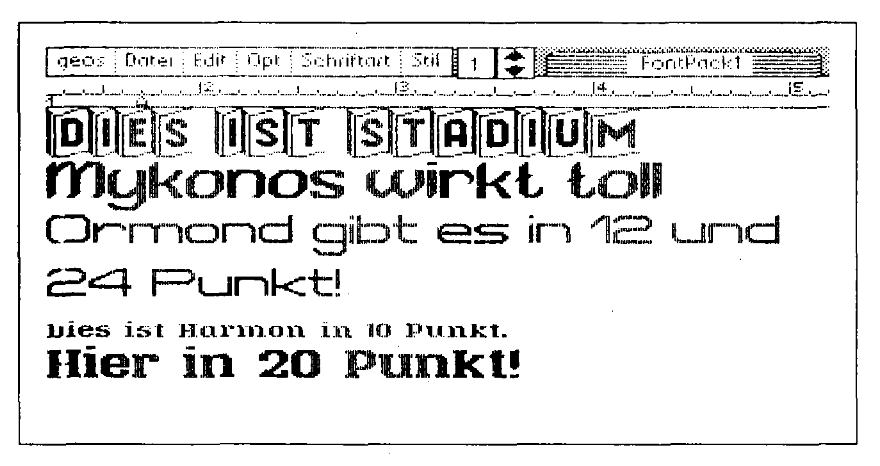


Bild 1.8.7.2: Neue Schriftarten: Stadium, Mykonos, Ormond, Harmon

Auch Stadium ist für Überschriften sehr gut geeignet. Es erinnert stark an Fahnen. Mykonos wirkt dagegen geheimnisvoll und ist wohl gut für Ihren nächsten Horror-Roman geeignet. Ormond sieht sehr utopisch aus. Harmon wirkt im Gegensatz zu Elmwood sehr harmonisch.

1.8.8 GEOPAINT

Inhalt der Arbeitsdisketten

Ihre Arbeitsdiskette sollte folgende Programme enthalten. So haben Sie immer alles, was sie brauchen, auf einer Diskette.

```
Desktop
Geowrite
Geopaint
Voreinstellung
Druckertreiber
Schriftfonts cory ge
             Dwinelle ge
             California ge
             Roma ge
```

Voreinstellung

Bei der Arbeit mit GEOPAINT ist es besonders beim punktgenauen Zeichnen ratsam, die Anfangsgeschwindkeit des Mauspfeils zu reduzieren, da man sonst ständig am gewünschten Punkt vorbeifährt und sich somit unnötige Arbeit macht.

Aber man kann das Programm VOREINSTELLUNG ja auch aus GEOPAINT und GEOWRITE laden, um die Einstellungen zu korrigieren. Nehmen Sie sich ruhig Zeit, die Uhr und das Datum richtig zu stellen. Denken Sie daran, das jede Grafik und jeder Text mit den eingestellten Daten gespeichert werden. Sie haben hiermit jederzeit einen Überblick, was Sie wann gemacht haben.

Visitenkarten

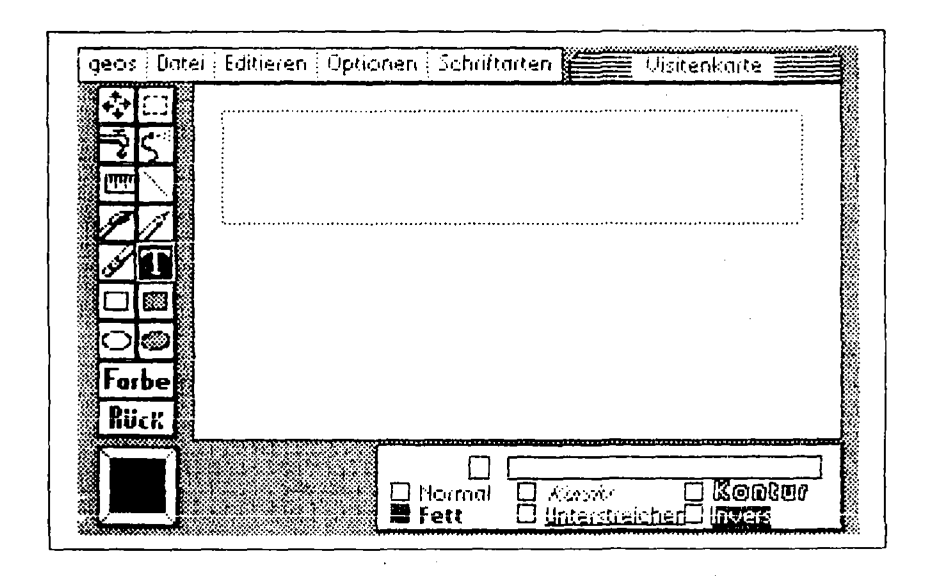
Auch hier habe ich Ihnen noch einmal die einzelnen Arbeitsschritte zur Herstellung einer Visitenkarte aufgeführt.

► GEOPAINT-Icon doppelklicken	→ GEOPAINT wird geladen, Auswahlfenster erscheint.
► Option NEUES DOKUMENT ERSTELLEN anwählen Name "VISITENKARTE" eingeben und mit <Return> bestätigen	→ Frage nach Dokumentname. → GEOPAINT-Zeichenblatt erscheint auf dem Bildschirm.
► Werkzeugleiste/Text anwählen	→ Hilfsmenü Style erscheint unten links.
► Menüleiste/Schriftarten/Roma/ 24 Punkte anwählen	
► Hilfsmenü/Fett anwählen Den Cursor bis fast in die obere linke Ecke bewegen klicken	→ Das Textfenster wird aktiviert.
► Textfenster mit dem Joystick nach rechts unten öffnen klicken Text eingeben "Manfred Müller"	→ Text-Cursor erscheint im Textfenster.

▶ **Werkzeugleiste/Linien anwählen**	→Hilfsmenü ändert sich von Style auf Maßeinheiten.
▶ **Cursor ca. 1 cm unter das M von Manfred bewegen klicken**	→Linienfunktion wird aktiv.
▶ **Waagerechte Linie mit Joystick bis an den rechten Rand ziehen**	
▶ **Werkzeugleiste/Text anwählen**	
▶ **Menüleiste/Schriftarten/ Roma/12 Punkte anwählen**	
▶ **Hilfsmenü/Fett anwählen Cursor in Bildausschnitt-mitte bewegen klicken**	Das Textfenster wird aktiviert.
▶ **Textfenster mit dem Joystick nach rechts unten öffnen klicken**	→Text-Cursor erscheint im Textfenster.
▶ **Text eingeben "Mühlenstraße 13" <Return> drücken, um einen Zeilenvorschub durchzuführen und in die nächste Zeile zu gelangen Text eingeben "4000 Düsseldorf" 2 mal <Return> (Zeilenvorschub), damit man eine Leerzeile erhält**	
▶ **Text eingeben "Telefon 0211/654321"**	
▶ **Werkzeugleiste/Rahmen anwählen**	→Hilfsmenü ändert sich von Style auf Editiermodus.
▶ **Hilfsmenü/Kopieren anwählen Cursor knapp links über die Linie positionieren klicken**	→Markierungsfunktion wird aktiv.

▶ **Mit dem Joystick den Rahmen so weit öffnen, bis Linie eng eingerahmt ist.** **klicken**	→Markierung ist fixiert.
▶ **Cursor in den Rahmen bewegen** **klicken**	→Die im Hilfsmenü gewählte Funktion ist aktiv, der Cursor ist hell gefärbt.
▶ **Mit dem Joystick Cursor zwischen Ort und Telefon bewegen** **klicken**	→Der markierte Rahmen folgt dem Cursor. →Die kopierte Linie ist fixiert.
▶ **Werkzeugleiste/Bleistift anwählen Den Bleistift mit dem Joystick ganz an den linken Rand setzen. (Es muß genau sein, da aus dem Bleistift sonst wieder der Mauspfeil wird.)** **klicken**	→Der Bleistift wird aktiv.
▶ **Bewegen Sie den Stift mit Joystick ganz nach oben. Wenn der Stift jetzt zum Mauspfeil wird, ändert es nichts an der Funktion. Bewegen Sie den inzwischen entstandenen Mauspfeil einmal um das ganze Arbeitsblatt herum. Der Bildausschnitt wird damit komplett eingerahmt.**	
▶ **Werkzeugleiste/ Bewegungspfeile anwählen Mit dem Joystick den Bildausschnitt so weit nach rechts verschieben, bis der Bildschirm wieder leer ist.**	
▶ **Werkzeugleiste/Text anwählen**	
▶ **Menüleiste/Schriftarten/ LW Cal/18 Punkte anwählen**	

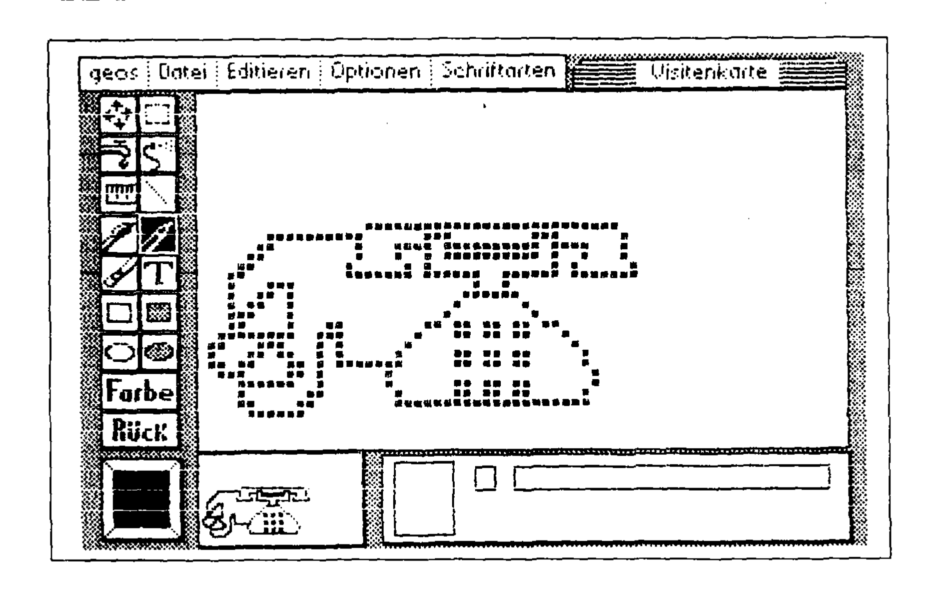
▶ **Hilfsmenü/Fett/Kursiv/ Kontur anwählen klicken**	→Das Textfenster wird aktiviert.
▶ **Textfenster von oben links anfangend über ganze Breite öffnen klicken Text eingeben "Manfred Müller"**	→Text-Cursor erscheint im Textfenster. →Zwischen Vor- und Nachname 3 Leerzeichen.



▶ **Werkzeugleiste/Text anwählen**	
▶ **Menüleiste/Schriftarten/ LW Cal/12 Punkte anwählen**	
▶ **Hilfsmenü/normal/Fett anwählen**	→Nach dem Anwählen des Punktes "Normal" werden alle vorher getroffenen Einstellungen gelöscht.
▶ **Cursor unter das M von Manfred positionieren klicken**	→Das Textfenster wird aktiviert.

▶ **Textfenster über die ganze Breite öffnen klicken**	→Text-Cursor erscheint im Textfenster.
▶ **Text eingeben "Versicherungen aller Art"**	
▶ **Werkzeugleiste/Text anwählen**	
▶ **Menüleiste/Schriftarten/ LW Cal/12 Punkte anwählen**	
▶ **Hilfsmenü/Fett/Kursiv anwählen Cursor in die Bildmitte bewegen klicken**	→Das *Textfenster* wird aktiviert.
▶ **Textfenster bis in die rechte untere Ecke öffnen klicken**	→Text-Cursor erscheint im Textfenster.
▶ **Text eingeben "Mühlenstraße 13" <Return> drücken zum *Zeilenvorschub* Text eingeben "4000 Düsseldorf" Zweimal <Return> drücken, um eine Leerzeile zu erhalten**	
▶ **Text eingeben "0211/654321"**	
▶ **Werkzeugleiste/ Linien anwählen Zwei Linien ziehen über und unter dem Text** VERSICHERUNGEN ALLER ART	
▶ **Werkzeugleiste/Bleistift anwählen**	
▶ **Menüleiste/Optionen/ Einzelpunkt anwählen**	→Einzelpunktrahmen erscheint oben links im Bild.
▶ **Den Rahmen mit dem Joystick vor die *Telefonnummer* bewegen. klicken**	→Der Inhalt des Rahmens wird im Bildausschnittfenster gezeigt.

▶ **Mit dem Stift ein Telefon zeichnen** → Hier sind der eigenen Phantasie keine Grenzen gesetzt.



▶ **Menüleiste/Optionen/ Normalmodus anwählen** → Bild wird wieder in Normalgröße gezeigt

▶ **Werkzeugleiste/ Bleistift anwählen Den Bleistift mit dem Joystick an den linken Rand setzen. (Es muß genau sein, da aus dem Bleistift sonst wieder der Mauspfeil wird.) klicken** → Der Bleistift wird aktiv.

▶ **Bewegen Sie den Stift mit dem Joystick ganz nach oben. Wenn der Stift jetzt zum Mauspfeil wird, ändert es nichts an der Funktion.**

▶ **Bewegen Sie den entstandenen Mauspfeil einmal um das ganze Arbeitsblatt herum. Der Bildausschnitt wird damit komplett eingerahmt.**	
▶ **Werkzeugleiste/ Bewegungspfeile anwählen Bildausschnitt ganz nach links verschieben, bis die erste Karte im Bild ist Bildausschnitt nach unten verschieben, bis der Bildausschnitt ganz frei ist**	
▶ **Werkzeugleiste/Text anwählen**	
▶ **Menüleiste/Schriftarten/ Roma/24 Punkte**	
▶ **Hilfsmenü/Fett anwählen Cursor oben links postieren klicken**	→Das Textfenster wird aktiviert.
▶ **Textfenster von oben links über die ganze Breite öffnen klicken Text eingeben "Manfred Müller"**	→Text-Cursor erscheint im Textfenster.
▶ **Werkzeugleiste/Text anwählen**	
▶ **Menüleiste/Schriftarten/ LW Cal/10 Punkte**	
▶ **Hilfsmenü/Normal anwählen Cursor rechts neben das Feld FARBE der Werkzeugleiste bewegen klicken**	→Das Textfenster wird aktiviert.

▶ **Textfenster in waagerechter Verlängerung des FARBE-Feldes der Werkzeugleiste über die ganze Breite nach unten rechts öffnen klicken**	→Text-Cursor erscheint im Textfenster.
▶ **Text eingeben: "Mühlenstraße 13 4000 Düsseldorf Tel.0211/654321"**	
▶ **Werkzeugleiste/Pinsel anwählen Mit dem Joystick den Pinsel unter M des Vornamens bewegen klicken**	→Pinselfunktion ist aktiv.
▶ **Waagerechte Linie nach rechts ziehen Auf beiden Seiten ca. 2 Pinselbreiten freilassen Eine zweite identische Linie über die Adresse ziehen**	
▶ **Werkzeugleiste/Text anwählen Cursor über der unteren Linie in Höhe des Wortes Tel. postieren klicken**	→Das Textfenster wird aktiviert.
▶ **Textfenster nach rechts über die ganze Breite und nach unten bis knapp über die Linie öffnen klicken Text eingeben "Optische Geräte"**	→Text-Cursor erscheint im Textfenster.
▶ **Werkzeugleiste/Kreis anwählen Cursor zwischen die beiden Linien in Höhe des Buchstabens A vom Vornamen positionieren klicken**	→Kreisfunktion ist aktiv.

► **Mit dem Joystick den Kreis nach rechts öffnen, bis der Kreis fast an die Linien stößt. klicken**	→Der Kreis ist fixiert.
► **Den Cursor jetzt bis unter den Buchstaben N des Vornamens wieder nach links bewegen. klicken**	→Kreisfunktion ist aktiv.
► **Mit dem Joystick den Kreis nach rechts öffnen, bis der Kreis fast an den äußeren Kreis stößt. klicken**	→Der Kreis ist fixiert.
► **Cursor zwischen die beiden Linien in Höhe des Leerzeichens zwischen Vorname und Nachname positionieren klicken**	→Kreisfunktion ist aktiv.
► **Mit dem Joystick den Kreis nach links öffnen, bis der Kreis fast an die Linien stößt. klicken**	→Der Kreis ist fixiert.
► **Den Cursor jetzt bis unter den Buchstaben D des Vornamens wieder nach rechts bewegen. klicken**	→Kreisfunktion ist aktiv.
► **Mit dem Joystick den Kreis nach links öffnen, bis der Kreis fast an den äußeren Kreis stößt. klicken**	→Der Kreis ist fixiert.
► **Werkzeugleiste/Bleistift anwählen Den Bleistift mit dem Joystick ganz an den linken Rand setzen. (Es muß genau sein, da aus dem Bleistift sonst der Mauspfeil wird.) klicken**	→Der Bleistift wird aktiv.

▶ **Bewegen Sie den Stift mit Joystick ganz nach oben. Wenn der Stift jetzt zum Mauspfeil wird, ändert es nichts an der Funktion. Bewegen Sie den inzwischen entstandenen Mauspfeil einmal um das ganze Arbeitsblatt herum. Der Bildausschnitt wird damit komplett eingerahmt.**	
▶ **Werkzeugleiste/ Bewegungspfeile anwählen Bildausschnitt nach rechts verschieben, bis der Bildausschnitt ganz frei ist**	
▶ **Werkzeugleiste/Text anwählen**	
▶ **Menüleiste/Schriftarten/ Cory/24 Punkte anwählen**	
▶ **Hilfsmenü/Fett anwählen Den Cursor bis fast in die obere linke Ecke bewegen klicken**	→Das Textfenster wird aktiviert.
▶ **Textfenster mit dem Joystick nach rechts unten öffnen klicken Text eingeben "Manfred Müller"**	→Text-Cursor erscheint im Textfenster.
▶ **Menüleiste/Schriftarten/ Cory/12 Punkte anwählen**	
▶ **Hilfsmenü/Fett anwählen Cursor unter das M von Manfred positionieren klicken**	→Das Textfenster wird aktiviert.
▶ **Textfenster über ganze Breite öffnen klicken Text eingeben "Gebrauchtwagenhandel"**	→Text-Cursor erscheint im Textfenster.

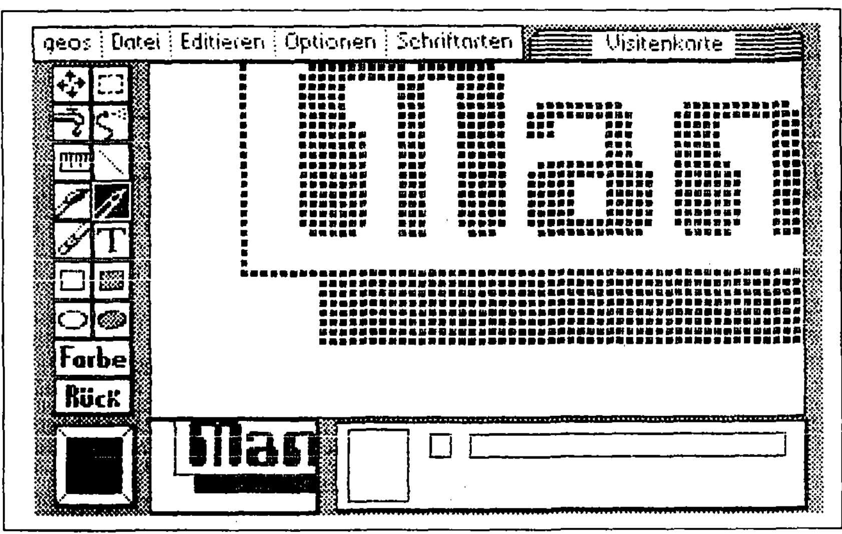
▶ **Werkzeugleiste/Text anwählen**	
▶ **Menüleiste/Schriftarten/ Cory/12 Punkte anwählen**	
▶ **Hilfsmenü/Normal anwählen Cursor in die Bildmitte bewegen klicken**	→Das Textfenster wird aktiviert.
▶ **Textfenster bis in rechte untere Ecke öffnen klicken Text eingeben "Mühlenstraße 13" <Return> drücken zum Zeilenvorschub Text eingeben "4000 Düsseldorf" Zweimal <Return> drücken, um eine Leerzeile zu erhalten**	→Text-Cursor erscheint im Textfenster.
▶ **Text eingeben "0211/654321"**	
▶ **Werkzeugleiste/Rechtecke anwählen Cursor links über Namen bewegen klicken**	→Rechteckfunktion ist aktiv.
▶ **Mit dem Joystick das Rechteck bis rechts unter den Namen öffnen Klicken Das gleiche mit den restlichen Texten**	→Rechteck ist fixiert.

▶ **Werkzeugleiste/Pinsel anwählen**	
▶ **Menüleiste/Optionen/ Pinsel wechseln Den zweiten Rechteckpinsel von links wählen.**	
▶ **Menüleiste/Optionen/ Einzelpunkt anwählen Mit dem Joystick Auswahlfenster**	→Jetzt kann Feinarbeit gemacht werden.

auf die erste Ecke bewegen
klicken

→Im Bildausschnittfenster wird der gewählte Bereich gezeigt.



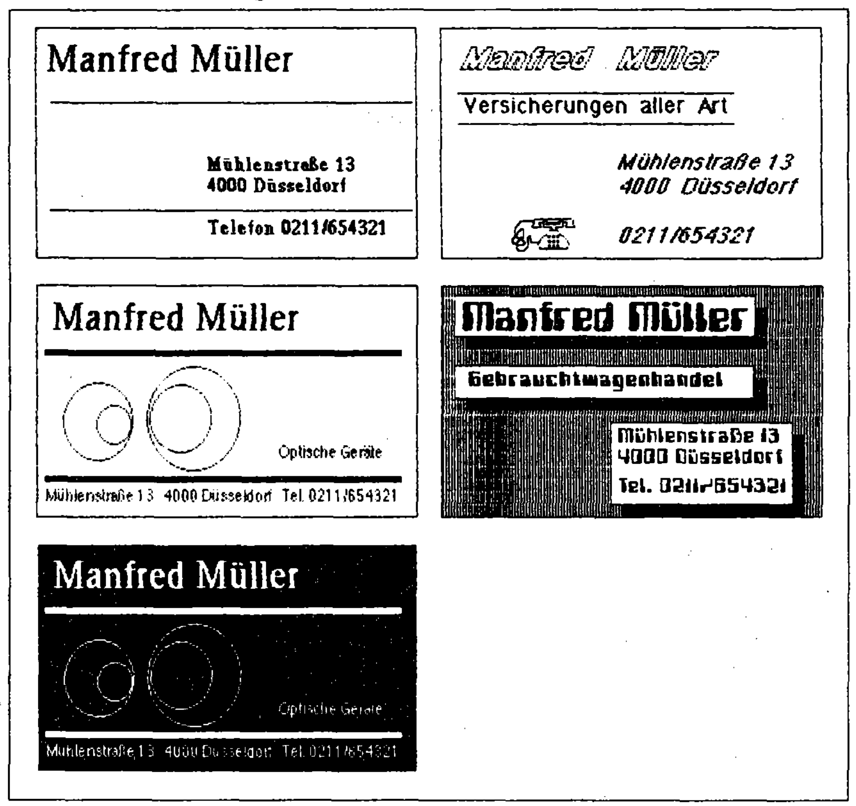


Sollte die Ecke bei Ihnen nicht so aussehen, kann sie jetzt nachbearbeitet werden. Hierzu:

▶**Werkzeugleiste/Bleistift anwählen**	→Stift ist aktiv.
▶**Werkzeugleiste/ Bewegungspfeile anwählen Mit dem Joystick den Auswahlrahmen zur nächsten Ecke oder sonstigen Stelle, welche nachgebessert werden muß, bewegen**	→Wenn alles nachgebessert ist:
▶**Menüleiste/Optionen/ Normalmodus anwählen**	
▶**Werkzeugleiste/Bleistift anwählen Den Bleistift mit dem Joystick ganz an den linken Rand setzen. (Es muß genau sein, da aus dem Bleistift sonst der Mauspfeil wird.) klicken**	→Der Bleistift wird aktiv.
▶**Bewegen Sie den Stift mit dem Joystick ganz nach oben. Wenn der Stift zum Mauspfeil wird, ändert es nichts an der Funktion. Bewegen Sie den entstandenen Mauspfeil einmal um das ganze Arbeitsblatt herum. Bildausschnitt wird damit komplett eingerahmt.**	
▶**Werkzeugleiste/ Wasserhahn anwählen Cursor mit dem Joystick auf das Feld mit der akuellen Füllfarbe unten links bewegen. klicken**	→Zur Auswahl stehende Füllarben werden angezeigt.
▶**Durch Anklicken die sechste Farbe von rechts in der oberen Reihe wählen**	→Aktuelle Füllfarbe wird angezeigt.

▶ **Den Cursor unten links ins Bildausschnittfenster bewegen klicken**	→ Das Feld wird mit gewählter Farbe gefüllt.
▶ **Werkzeugleiste/ Bewegungspfeile anwählen Bildausschnittfenster ganz nach links verschieben, bis die vorherige Karte wieder im Fenster ist klicken**	→ Bewegungspfeile werden deaktiviert.
▶ **Werkzeugleiste/Rahmen anwählen**	
▶ **Werkzeugleiste/ Rahmen doppelklicken**	→ Ganzer Bildausschnitt wird gerahmt.
▶ **Menüleiste/Editieren/ Kopieren anwählen**	→ Gerahmter Bildausschnitt wird zwischengespeichert.
▶ **Werkzeugleiste/ Bewegungspfeile anwählen; Bildausschnitt nach unten bewegen, bis Fenster ganz frei ist. klicken**	→ Bewegungspfeile werden deaktiviert.
▶ **Werkzeugleiste/Rahmen anwählen**	
▶ **Werkzeugleiste/ Rahmen doppelklicken**	→ Ganzer Bildausschnitt wird gerahmt.
▶ **Menüleiste/Editieren/ Einkleben anwählen**	→ Der eben zwischengespeicherte Bildausschnitt wird in das markierte Feld eingeklebt.
▶ **Werkzeugleiste/Rahmen anwählen**	
▶ **Werkzeugleiste/ Rahmen doppelklicken**	→ Ganzer Bildausschnitt wird gerahmt.
▶ **Hilfsmenü/Invertieren anwählen**	→ Der gerahmte Bereich wird invertiert dargestellt.

Hier nun noch einige Visitenkarten:



Etiketten

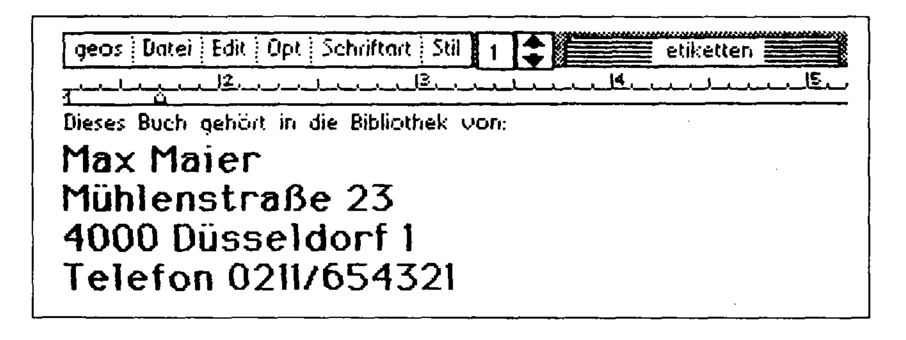
Sie benötigen sicher manchmal Etiketten mit Ihrem Namen, um z.B. Bücher zu kennzeichnen, welche man ab und zu an Freunde und Bekannte ausleiht. Zu diesem Zweck habe ich mir mit GEOWRITE Etiketten erstellt. Laden Sie dazu als erstes GEOWRITE. Als Dokumentname tippen Sie ETIKETTEN ein. Nachdem das Programm geladen ist, haben Sie das leere Blatt auf dem Bildschirm. Der Cursor steht in der linken oberen Ecke. Wir geben hier die erste Zeile Text ein.

"Dieses Buch gehört in die Bibliothek von:"

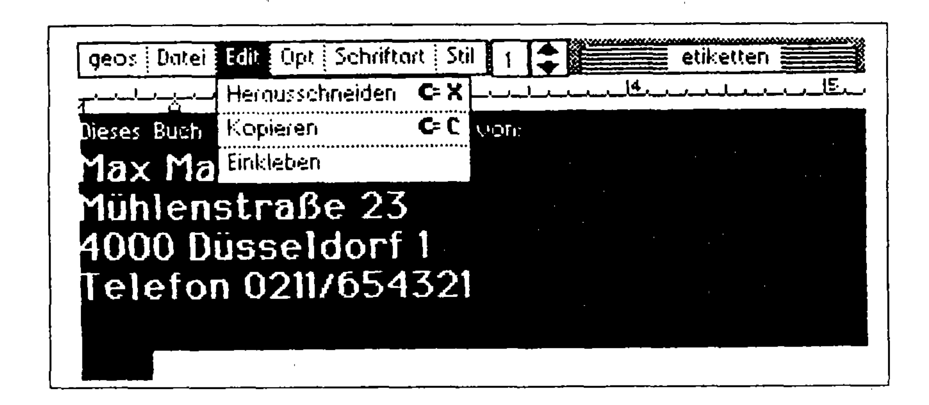
Der Text wird in der Schriftart BSW 9 Punkt geschrieben, da diese Schriftart voreingestellt ist. Nachdem wir den Cursor durch Drücken von <Return> in die nächste Zeile bewegt haben, wählen wir nun eine größere Schriftart aus. In diesem Beispiel habe ich CALIFORNIA 14 Punkt gewählt. Wir geben jetzt unseren Namen und unsere Adresse ein. Sie können selbstverständlich Ihren eigenen Namen nehmen:

```
"Max Maier"
"Mühlerstraße 23"
"4000 Düsseldorf 1"
"Telefon 0211/654321"
```

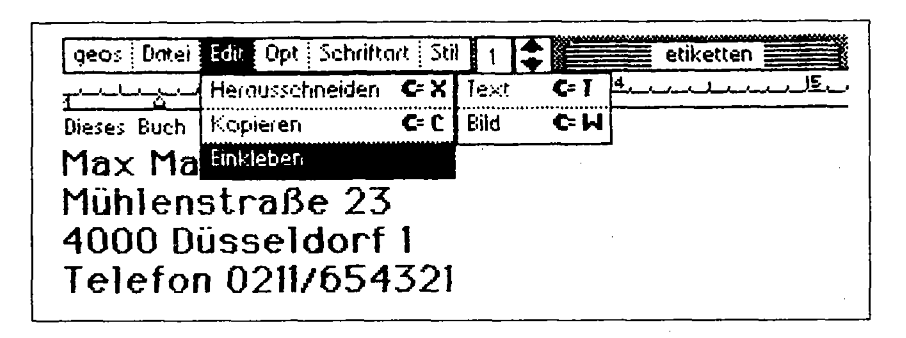
Jetzt müßte es auf Ihrem Bildschirm eigentlich so aussehen, natürlich mit Ihrem Namen.



Danach geben wir in die nächsten beiden Zeilen jeweils einige Leerzeichen ein, welche wir auch mit <Return> abschließen. Diese Leerzeichen (Leerzeilen) können wir nutzen, um unseren Ausdruck den verwendeten Etiketten größenmäßig anzupassen, da wir ja kein Etikettenmaß eingeben können, sondern wir nur in "Pixelmaß" messen können. Um den Abstand zwischen den einzelnen Adressen jedoch genau einstellen zu können, benötigen wir noch mindestens eine weitere Adresse. Selbstverständlich brauchen wir diese jetzt nicht einzutippen, sondern nutzen hier die Möglichkeit, Text auszuschneiden und zwischenzuspeichern. Markieren Sie hierzu bitte den gesamten Text incl. der beiden Leerzeilen, und wählen Sie unter der Option EDIT die Funktion KOPIEREN.



Nachdem der markierte Text gespeichert ist, bewegen Sie den Cursor unter die 2. Leerzeile an den Zeilenanfang und wählen wiederum unter der Option EDIT den Punkt EINKLEBEN und dann TEXT. Sie sehen, der eben gespeicherte Text wird von der Cursor-Position an auf den Bildschirm ausgegeben.



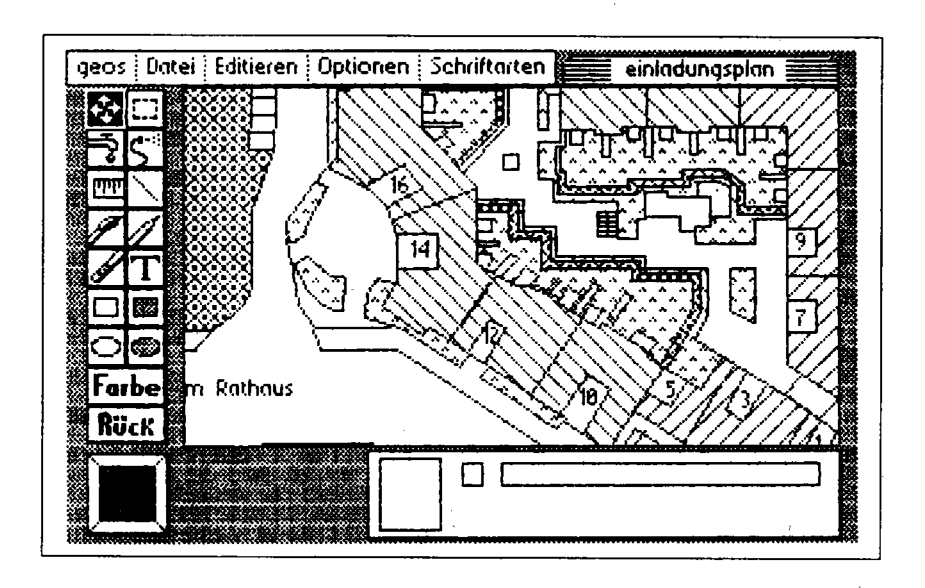
Damit ist der Text jedoch nicht verschwunden, sondern bleibt so lange zwischengespeichert, bis ein neuer Text ausgeschnitten oder kopiert wird. Bewegen Sie den Cursor also nochmals an das Textende unter die beiden Leerzeilen, und wählen Sie TEXT EINKLEBEN. Es erscheint wieder die gespeicherte Adresse. Jetzt sollten Sie den ersten Probeausdruck machen. Da Etiketten einiges mehr kosten als normales Papier, machen Sie Ihre Probeausdrucke ruhig auf Endlospapier und halten es neben Ihre Etiketten, um zu prüfen, ob die Adressen den richtigen Abstand

voneinander haben. Sollte der Abstand nicht korrekt sein, haben Sie die Möglichkeit, diesen zu vergrößern oder zu verkleinern, indem Sie die Leerzeilen markieren und eine andere Schriftart oder -größe wählen. Sobald der Abstand Ihren Etiketten entspricht, kleben Sie noch so viele Etiketten untereinander, bis die erste Seite voll ist. Sie können jetzt Etiketten für all Ihre Bücher drucken. Sie können den Text aber auch erst mal speichern, und wann immer Sie wollen, direkt vom Desktop aus Ihre Etiketten ausdrucken.

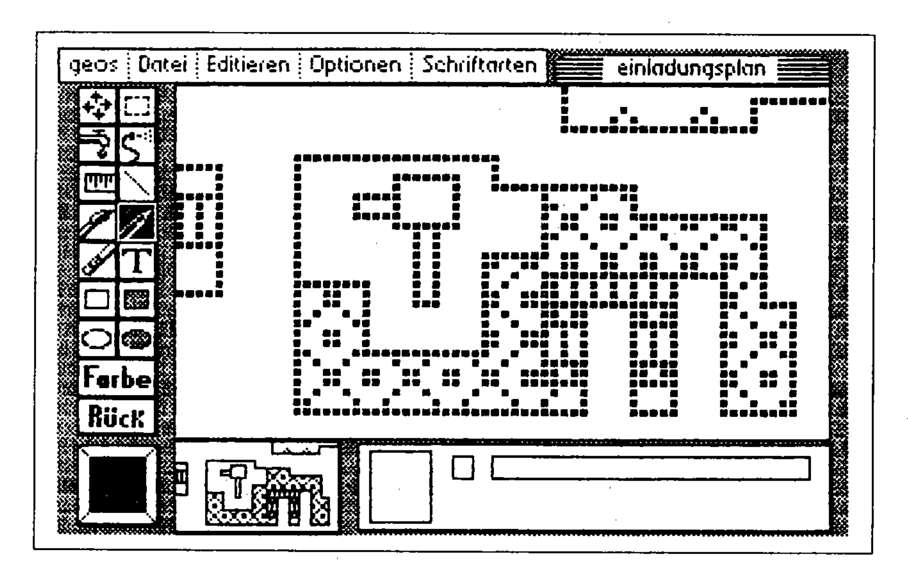
Dieses Buch gehört in die Bibliothek von:
Max Maier
Mühlenstraße 23
4000 Düsseldorf 1
Telefon 0211/654321

Stadtplan

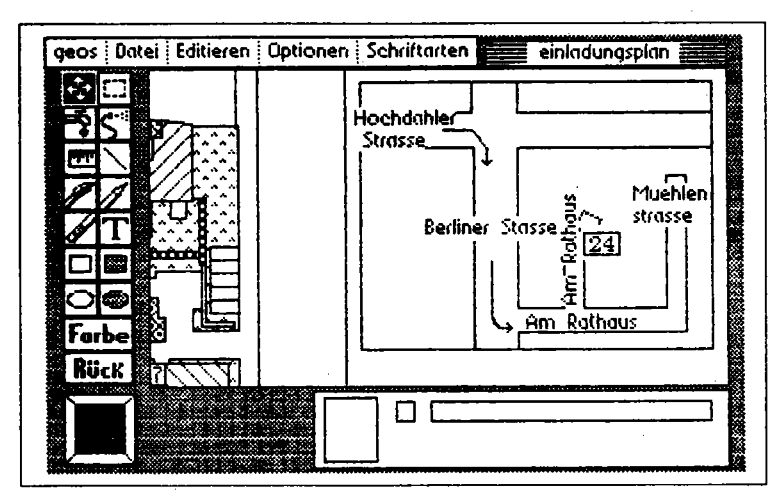
Diese Anwendung wollte ich eigentlich gar nicht mit ins Buch bringen, da hierfür ein derart hoher Zeitaufwand notwendig ist,daß es viel zu lange dauern würde, bis man alles beschrieben hätte. Um Ihnen jedoch zu zeigen, was mit GEOPAINT machbar ist, werde ich wenigstens kurz meine Vorgehensweise zur Erstellung eines - von den Proportionen her korrekt gezeichneten - Stadtplanausschnitts schildern. Gezeichnet habe ich diesen Ausschnitt, als wir in eine neue Wohnung eingezogen sind, und jeder, der uns besuchen wollte, nach einer genauen Wegbeschreibung fragte. Ich habe über den Originalplan eine dünne Klarsichtfolie gelegt und die Umrisse übernommen. Diese Folie habe ich dann auf den Bildschirm gelegt (wenn die Folie dünn genug ist, haftet diese sehr gut) und erst die Umrisse mit GEOPAINT gezeichnet. Die Folie habe ich mit dem Verschieben des Bildausschnitts identisch mitverschoben. Nachdem die Umrisse fertig waren, wurde ein Bildausschnitt nach dem anderen dem Originalplan angepaßt.



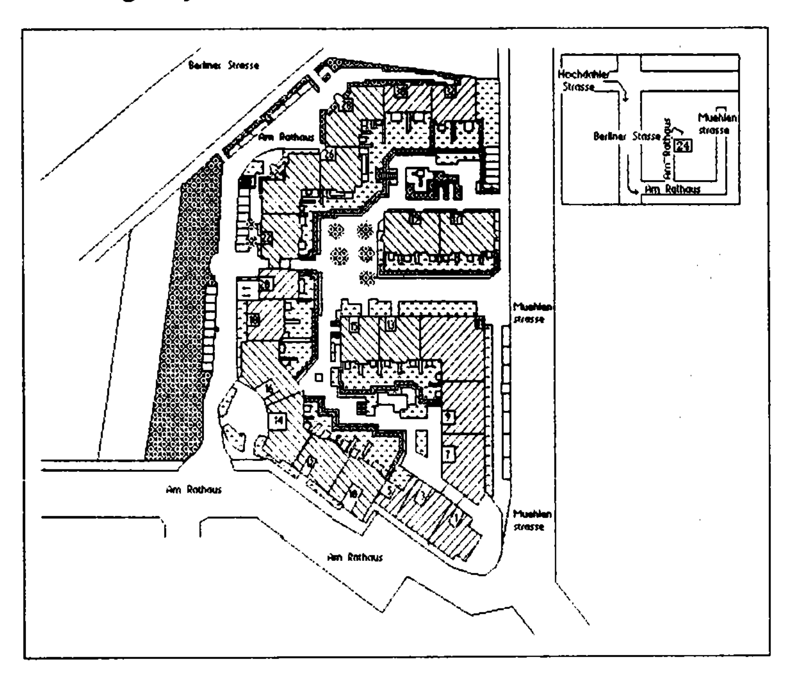
Vieles ließe sich nur im Einzelpunktmodus erstellen.



An eine freie Stelle habe ich dann noch eine kleine Wegbeschreibung eingezeichnet.



Nach stundenlanger Arbeit sah der Plan dann auch tatsächlich wie der Originalplan aus.



Sicher ist das Zeichnen eines Stadtplans keine typische Anwendung für dieses Programm, aber es erstaunt mich immer wieder, wie einfach und vielfältig, wenn auch manchmal zeitaufwendig die Arbeit mit GEOPAINT sein kann.

2. Was man mit dem C64 alles machen kann

Mittlerweile gibt es für den Commodore 64 ein fast unüberschaubares Angebot an Software- und Hardware-Erweiterungen. Es ist schon erstaunlich, was die Programmierer in der Zwischenzeit aus dem 64'er herausgeholt haben. Ein gutes Beispiel dafür dürfte die Benutzeroberfläche GEOS sein.

Natürlich kann man von einer Commodore-64-Anlage nicht die Leistung einer mehrere tausend Mark teuren Personalcomputer-Anlage erwarten. Für private oder semiprofessionelle Anwendungen reicht der Commodore 64 in der Regel aber allemal aus. Man benötigt eben nur die richtige Software. Da die Software-Preise im 64'er-Bereich in der Zwischenzeit ein - zumindest für den Anwender - erfreulich niedriges Niveau erreicht haben, dürfte es Ihnen nicht schwerfallen, Ihre individuelle Software-Sammlung zusammenzustellen. Während man im PC-Bereich beispielsweise für eine Textverarbeitung, die wirklich allen Anforderungen gerecht wird, deutlich über tausend Mark hinblättern muß, gibt es gute Textverarbeitungen für den Commodore 64 schon für unter 100 Mark.

Was erwartet Sie nun in diesem Kapitel? Wie die Überschrift schon erkennen läßt, möchte ich Ihnen im folgenden zeigen, was man mit dem 64'er - außer ihn selbst zu programmieren - alles anfangen kann. Neben ernsthaften Anwendungen, wie Daten- und Textverarbeitung, kommt dabei natürlich auch das Spielen nicht zu kurz.

Ich möchte Ihnen hier allerdings keine kompletten Marktübersichten oder eine umfangreiche Sammlung von Testberichten präsentieren. Beide wären schon kurz nach Erscheinen dieses Buches überholt und veraltet. Was ich vielmehr möchte, ist es, Ihnen zu erklären, über welche grundsätzlichen Eigenschaften aber auch Unterschiede die verschiedenen Soft- und Hardware-Produkte verfügen.

Worin liegt beispielsweise der Unterschied zwischen einer einfachen Textverarbeitung und einem Desktop-Publishing-Programm? Sollte man seine Daten lieber mit einer Dateiverwaltung oder doch gleich mit Hilfe einer Datenbank verwalten? Welche Arten von Spiele-Software gibt es für den Commodore 64? Lohnt es sich, den Commodore 64 als Lernhilfe einzusetzen? All das sind Fragen, die ich Ihnen in diesem Kapitel beantworten möchte.

Natürlich stelle ich Ihnen dazu auch konkrete Produkte vor. Schließlich soll das Ganze ja nicht in graue Theorie ausarten. Alle vorgestellten Programme und Hardwareerweiterungen sind im Fachhandel oder direkt beim Hersteller erhältlich. Ein Anbieterverzeichnis finden Sie im Anhang.

Bevor wir mit unserem Streifzug durch das Produktangebot für den Commodore 64 beginnen, noch ein ernstes Wort: Wer unter zum Teil erheblichen Arbeitsaufwand Software für den 64'er entwickelt, hat auch ein berechtigtes Interesse daran, durch den Verkauf dieser Software etwas zu verdienen. Im Gegensatz zu sogenannten Public-Domain-Programmen, die frei kopiert werden dürfen, sollte man kommerzielle Software daher - schon in seinem eigenen Interesse - unbedingt kaufen und nicht raubkopieren.

2.1 Textverarbeitung

Beginnen wir mit der wohl häufigsten Computeranwendung, zumindest im professionellen Bereich: der Textverarbeitung. Egal, ob es sich nur um einen kurzen Brief oder ein komplettes Buch handelt, mit einem Computer läßt sich jede Art von Text "verarbeiten". Wie komfortabel und in welcher Qualität, das hängt in erster Linie von dem verwendeten Textverarbeitungsprogramm ab. Mindestens genau so wichtig ist aber ein guter Drucker. Während man bei Schreibmaschinen schon für vielleicht 300 bis 400 Mark eine brauchbare Druckqualität erhält, muß man bei Computerdruckern schon knapp 800 Mark anlegen, um einen Drucker mit "Korrespondenz-Qualität", auch NLQ (Near Letter Quality)- oder LQ-Qualität genannt, zu bekommen.

Eine nicht zu unterschätzende Investition! Denn das beste Textverarbeitungsprogramm nützt Ihnen nichts, wenn die Texte dann in miserabler Qualität aus dem Drucker kommen. Dies gilt natürlich insbesondere dann, wenn Sie die Texte, beispielsweise Briefe, an andere weitergeben wollen.

Manch einer, der bisher häufig mit der Schreibmaschine gearbeitet hat und damit auch ganz gut zurecht gekommen ist, wird sich nun vielleicht fragen, ob es sich überhaupt lohnt, auf den Commodore 64 als neue "Schreibmaschine" umzusteigen. Worin liegen die Vorteile einer Textverarbeitung mit dem Computer?

Als vielleicht wichtigstes Stichwort möchte ich nur nennen: Tippfehler! Auch wer das Schreibmaschinenschreiben perfekt beherrscht, wird sich mitunter vertippen und ist dann oft gezwungen, eine komplette Seite neu zu schreiben. Und wem ist es nicht schon passiert, daß ein Brieftext unbedingt auf eine DIN-A-4-Seite passen sollte, und beim Schreiben des letzten Absatzes stellt man dann fest, daß es doch nicht ganz reicht. Auch hier hilft meist nur ein komplettes Neutippen des Textes.

Bei der Textverarbeitung mit dem Computer ist das grundsätzlich anders. Wenn Sie mit dem Commodore 64 und einem Textverarbeitungsprogramm einen Text schreiben, dann wird dieser zunächst nur im internen Speicher des Rechners abgelegt und jeweils ein Ausschnitt davon am Bildschirm angezeigt. Während sich der Text im Rechner befindet, können Sie ihn beliebig verändern, also zum Beispiel Tippfehler korrigieren, etwas ergänzen aber auch ganze Absätze vertauschen.

Erst, wenn der Text genauso ausschaut, wie Sie ihn haben wollten, bringen Sie ihn zu Papier. Und wenn Sie dann auf dem Ausdruck noch einen Fehler entdecken sollten, macht das natürlich überhaupt nichts. Schließlich haben Sie den Text ja noch im Rechner und können ihn nach Durchführung der Korrekturen jederzeit noch einmal ausdrucken.

Das Ausdrucken mit dem Drucker ist aber nicht die einzige Möglichkeit, einen Text dauerhaft zu "speichern". (Der Text im Rechnerspeicher geht ja nach dem Ausschalten des Rechners

verloren!) Zusätzlich können Sie den Text auch noch auf einer Kassette (wenn Sie über eine Datasette verfügen) oder besser auf einer Diskette (falls Sie eine Floppy Ihr eigen nennen) abspeichern. Das ist besonders dann hilfreich, wenn Sie einen Text, etwa eine Stellenbewerbung, mehrfach verwenden möchten. Aber auch bei längeren, nur einmal benötigten Texten ist es sehr empfehlenswert, diese zwischendurch immer wieder einmal abzuspeichern, um einem plötzlichen Stromausfall zu begegnen.

Um mit dem Commodore 64 sinnvoll Textverarbeitung betreiben zu können, benötigt man also neben einem guten Drucker auch ein dauerhaftes Speichermedium, am besten ein Diskettenlaufwerk. Eine Datasette ist nur sehr bedingt zu empfehlen. Allein schon deshalb, weil die meisten Textverarbeitungsprogramme nur auf Diskette ausgeliefert werden!

Und damit wären wir auch schon beim wichtigsten: dem Textverarbeitungsprogramm. Welches Textverarbeitungsprogramm für Sie am besten geeignet ist, hängt in erster Linie davon ab, welche Art von Texten Sie verarbeiten wollen. Soll es beispielsweise nur gelegentlich ein kurzer Brief sein, oder wollen Sie auch größere Dokumente, etwa Referate, mit dem Computer verarbeiten? Vielleicht haben Sie ja auch regelmäßig größere Mengen an sogenannten Serienbriefen, etwa zur Vereinsverwaltung, zu verarbeiten. Auch dafür gibt es bei den meisten Programmen eine spezielle Funktion, die sogenannte "Mail-Merge-Funktion". Mehr dazu etwas weiter unten.

Aus dem Meer an Textverarbeitungen möchte ich drei Programme herausgreifen und Ihnen an diesen ausschnittsweise zeigen, über welche Fähigkeiten eine gute Textverarbeitung verfügen sollte: Vizawrite, Textomat Plus und GEOWRITE.

Vizawrite und Textomat Plus kosten jeweils unter 100,- Mark, GEOWRITE ist in Ihrem GEOS-Paket enthalten. GEOS 2.0 enthält allerdings eine wesentlich verbesserte Version von GEOWRITE.

Die drei Programme verfolgen jeweils recht unterschiedliche Konzeptionen, was den Programmablauf betrifft. Das fängt

schon bei der Bedienung an. Während man sich bei Vizawrite relativ viele Kommandos auswendig merken muß, da das Programm über keine Menüsteuerung verfügt, läßt sich Textomat Plus sehr komfortabel über Menüs bedienen, wenn auch nicht über sogenannte Pulldown-Menüs, wie bei GEOS. GEOWRITE ist natürlich voll in das GEOS-Bedienungskonzept integriert und bietet damit den größtmöglichen Bedienungskomfort.

Ein ganz wichtiger Punkt bei der Textverarbeitung ist die Darstellung des eingegebenen Textes am Bildschirm. Dabei haben alle Programme mit einem Handikap in der Bildschirmdarstellung des Commodore 64 zu kämpfen: Wie Sie in der Zwischenzeit sicher schon bemerkt haben, kann der Commodore 64 in einer Bildschirmzeile nur 40 Zeichen darstellen, eine Textzeile in einem Brief hat aber in der Regel eine Länge von 80 Zeichen! Die meisten Textverarbeitungsprogramme behelfen sich hier mit einem horizontalen Hin- und Herschieben des Textes, d.h., sobald Sie über die 40. Spalte hinausschreiben, wird der gesamte am Bildschirm sichtbare Text um mehrere Spalten nach links "gescrollt", wie der Fachmann sagt. Beginnt man dann eine neue Zeile, so wird der Text wieder nach rechts gescrollt.

Dieses ständige Hin- und Herschieben wirkt sich natürlich auf die Eingabegeschwindigkeit aus. Besonders bei GEOWRITE, das ja ohnehin nicht gerade schnell ist, macht sich das deutlich bemerkbar.

Ein weiterer Nachteil: Man kann sich nur schwer einen Überblick über den Gesamttext verschaffen. Einige Programme, beispielsweise Textomat Plus, bieten deshalb eine spezielle Übersichtsfunktion, bei der der Text dann, in 80-Zeichen-Darstellung verkleinert, aber noch gut lesbar dargestellt wird. Neben diesem grundsätzlichen Problem kann man zwei Arten der Bildschirmdarstellung unterscheiden:

Eine Gruppe von Textverarbeitungsprogrammen versucht, den Text am Bildschirm möglichst so darzustellen, wie er später auf dem Papier aussieht. In diesem Fall spricht man von "WYSIWYG". Diese merkwürdige Abkürzung steht für "What You See Is What You Get", also etwa "Was Du (am Bildschirm)

siehst, erhälst Du (auf dem Papier)". Das beste Beispiel für diese Darstellungsart ist GEOWRITE. Selbst aufwendige Schriftarten, wie etwa kursiv oder outline, werden direkt auf dem Bildschirm dargestellt.

So etwas kostet natürlich Zeit. Andere Textverarbeitungsprogramme begnügen sich deshalb mit einer relativ schlichten Bildschirmdarstellung und stellen bestimmte Formatierungen des Textes durch spezielle "Steuerzeichen" dar. Besonders wichtige Formatierungen, wie etwa die Zahl der Zeichen je Zeile oder die Anzahl der Zeilen je Seite, aber auch bestimmte Schriftarten, wie etwa Blocksatz oder Proportionalschrift, lassen sich zudem in der Regel zentral in einem Art "Formular" einstellen. Das hat den großen Vorteil, daß man die Formatierungen nicht nur blitzschnell ändern, sondern auch für mehrere Texte verwenden kann. So läßt sich beispielsweise sehr leicht ein Formular für Standardbriefe erstellen, zu dem man dann jeweils nur noch den reinen Brieftext hinzufügen muß.

Welche Art der Darstellung man bevorzugt, ist letztendlich Geschmackssache. Entscheidend ist ja nachher, wie der Text "Schwarz auf Weiß" gedruckt aussieht. Und das hängt entscheidend von der Qualität des Druckers ab. Welche Möglichkeiten hat man nun, den Text im Rechner zu bearbeiten?

Nachdem man die gewünschte Textbreite, d.h. die Anzahl der Zeichen je Zeile, und evtl. noch andere Werte eingestellt hat, kann man den Text eingeben. Bereits eingegebener Text läßt sich beliebig korrigieren und erweitern. Mit Hilfe des Cursors kann man an jede beliebige Stelle des Textes fahren, um dort etwas zu ändern.

Paßt am Ende einer Zeile ein angefangenes Wort nicht mehr ganz in die Zeile, so wird es automatisch ganz in die nächste Zeile "hinübergezogen". Diese nützliche Funktion nennt man "Word-Wrapping". Manche Programme bieten außerdem eine automatische Trennhilfe an, d.h., ein Wort, das nicht mehr ganz in eine Zeile paßt, wird automatisch getrennt. Wegen der komplizierten deutschen Grammatikregeln klappt das aber nicht immer korrekt.

Zum Standard jeder Textverarbeitung gehören mittlerweile auch Funktionen zum Löschen, Kopieren und Verschieben größerer Textblöcke. Damit kann man dann beispielsweise einen kompletten Textabsatz blitzschnell an eine andere Stelle des Textes verschieben oder mehrfach vorkommende Formulierungen nur einmal schreiben und dann kopieren.

Insbesondere bei längeren Texten sind auch Such- und Ersetzfunktionen sehr hilfreich. Stellt man zum Beispiel am Ende eines Textes fest, daß man den Namen einer Person, den man vielleicht 20 bis 30 Mal im Text verwendet hat, falsch geschrieben hat, so wäre es sehr mühsam, selbst auf die Suche zu gehen. Diese Arbeit kann man sich von der Textverarbeitung abnehmen lassen! Man ruft einfach die Such- und Ersetzfunktion auf und gibt den Suchbegriff (in diesem Fall den falsch geschriebenen Namen) und den Ersatzbegriff (den richtig geschriebenen Namen) an. Alles weitere erledigt das Programm.

Möchte man seinem Text ein professionelles Aussehen verleihen, so ist die richtige Formatierung sehr wichtig. Im einfachsten Fall wird der Text im sogenannten "Flattersatz" gedruckt, d.h. mit einem ungleichmäßigen rechten Rand wie bei der Schreibmaschine.

Beim sogenannten "Blocksatz" hat man einen ausgeglichenen rechten Rand. Die Leerräume zwischen den einzelnen Wörtern einer Zeile werden dazu von der Textverarbeitung mit zusätzlichen Leerzeichen aufgefüllt.

Das beste Schriftbild ergibt sich in Kombination mit der Proportionalschrift, wie sie auch manche moderne Schreibmaschinen bieten. Bei der Proportionalschrift werden die einzelnen Zeichen in unterschiedlichen Abständen gedruckt. Ein w beispielsweise ist ja wesentlich breiter als etwa ein i. Druckt man etwa das Wort wir in Normalschrift, so hat man zwischen dem w und dem i einen relativ großen Abstand. In Proportionalschrift wird das i daher dichter neben das w gesetzt; man erhält ein ausgeglicheneres Schriftbild.

Bei einer Schreibmaschine hat man in der Regel nur zwei Möglichkeiten, Text besonders hervorzuheben: Unterstreichen und Fettdruck. Ein moderner Matrixdrucker hat da schon wesentlich mehr zu bieten. Gehen wir die einzelnen Möglichkeiten einmal der Reihe nach durch:

Kursivschrift

Der Text wird schräggestellt gedruckt.

Fettdruck oder Doppeldruck

Der Text wird durch zweimaliges Drucken fett dargestellt.

Breitschrift

Der Text wird doppelt breit gedruckt, d.h., jedes Zeichen hat die doppelte Breite wie in Normalschrift.

Elite- und Pica-Schrift

Bei diesen beiden Schriftarten werden die Zeichen kleiner gedruckt, dadurch passen mehr Zeichen in eine Zeile. Sehr hilfreich beispielsweise beim Ausdrucken von Tabellen.

Vergrößerte Schrift oder doppelt hoher Druck

Der Text wird über die Höhe von zwei normalen Zeilen gedruckt. Interessant zum Beispiel bei Überschriften.

Reverser Druck

Der Text wird in Negativschrift, d.h. mit schwarzem Hintergrund gedruckt.

Sub- und Superscript

Mit Subscript bezeichnet man das Tiefstellen von Zeichen, mit Superscript das Hochstellen. Diese beiden Schriftarten eignen sich gut zum Ausdrucken von wissenschaftlichen Formeln.

Unterstreichfunktion

Der Text wird unterstrichen.

NLQ-Schrift oder Korrespondenzdruck

NLQ steht für "Near Letter Quality", also "Fast-Briefqualität". In Normalschrift sieht man bei einem Matrixdrucker deutlich, daß die Zeichen aus einzelnen Punkten zusammengesetzt sind. Das Schriftbild ist deshalb nicht besonders gut. In NLQ-Schrift wird jede Zeile zweimal gedruckt, das zweite Mal jedoch leicht versetzt zum ersten Mal. Dadurch wird der Zwischenraum zwischen den Punkten weitestgehend ausgefüllt. Das Schriftbild erreicht fast die Qualität einer Schreibmaschine.

Manche Schriftvariationen können auch gemischt genutzt werden, etwa Kursivschrift und Fettdruck.

Natürlich muß die Textverarbeitung dem Drucker irgendwie mitteilen, auf welche Art und Weise er den Text drucken soll. Dazu gibt es spezielle Steuersequenzen. Leider gibt es fast so viele Steuersequenzen für die einzelnen Druckerfunktionen wie es verschiedene Druckerfabrikate gibt. Ein Standard, der sich halbwegs durchgesetzt hat, ist der ESC/P-Standard des japanischen Druckerherstellers EPSON. Wenn Ihr Drucker also "EPSON-kompatibel" ist, d.h. die EPSON-Druckerbefehle "versteht", können Sie einigermaßen sicher sein, daß Ihre Textverarbeitung mit dem Drucker problemlos zusammenarbeitet. Für alle Fälle verfügen die meisten Textverarbeitungsprogramme über eine spezielle Druckeranpassungsfunktion, mit der man notfalls die erforderlichen Änderungen und Ergänzungen selbst durchführen kann.

Der Textspeicher innerhalb des Commodore 64 ist natürlich nicht unbegrenzt. Er schwankt bei den einzelnen Programmen zwischen 17000 und 34000 Zeichen. Das ist nicht besonders viel. Die meisten Textverarbeitungsprogramme bieten deshalb die Möglichkeit, mehrere Textdateien als einen zusammenhängenden Text aufzufassen. Mit dieser Methode kann man praktisch beliebig lange Texte bearbeiten. Beim Drucken wird dann zunächst

die eine Datei geladen und gedruckt und anschließend automatisch die nächste Textdatei geholt. Die Formatanweisungen innerhalb der ersten Textdatei gelten dann für den gesamten Text, so daß beispielsweise auch eine Seitennummerierung korrekt funktioniert.

Nachdem wir nun die Standardfunktionen, über die jede gute Textverarbeitung verfügen sollte, abgehakt haben, möchte ich noch auf einige spezielle Funktionen eingehen, über die nicht alle Programme verfügen.

Wer häufig mit Floskeln, wie "Sehr geehrte", oder "Mit freundlichen Grüßen" und ähnlichem, hantieren muß, wird sogenannte "Floskeltasten" begrüßen, wie sie zum Beispiel Textomat Plus bietet. Auf jede der insgesamt acht Floskeltasten, läßt sich ein beliebiger Text legen, der dann durch einen entsprechenden Tastendruck in den Text eingefügt wird.

Manche Programme bieten auch recht umfangreiche Dienstfunktionen zur Diskettenverwaltung an, etwa zum Löschen oder Umbenennen von Dateien oder zum Formatieren von Disketten. Das mag zwar vielleicht recht unscheinbar klingen, kann aber sehr wichtig sein, wenn Sie beispielsweise einen langen Text eingegeben haben und dann feststellen, daß Sie nur noch leere, unformatierte Disketten zum Speichern haben. Wenn die Textverarbeitung jetzt keine Funktion zur Diskettenformatierung anbietet, sieht es mit Ihrem Text sehr schlecht aus!

Eine überaus nützliche Funktion vieler Textverarbeitungsprogramme ist die sogenannte Serienbrief-Funktion, oft auch Mailmerge-Funktion genannt.

Nehmen wir einmal an, Sie möchten für einen Verein von vielleicht 50 Mitgliedern ein Rundschreiben verfassen, in dem die einzelnen Mitglieder zu einer Versammlung eingeladen werden sollen. Da das Ganze ja nicht zu unpersönlich werden soll, soll jedes Mitglied im Briefkopf und in der Anrede persönlich angesprochen werden.

Das einzige, worin sich die einzelnen Briefe unterscheiden, sind also die Anschrift und die Anrede "Sehr geehrte ...". Hat man nur eine Schreibmaschine zur Verfügung, dann schreibt man den Brief eben einmal, läßt dabei die Anschrift und die Anrede frei, kopiert das Ganze anschließend 50 Mal und fügt dann die einzelnen Adressen ein. Das macht nicht nur eine Menge Arbeit, man sieht den Briefen auch deutlich an, wie sie erstellt wurden. Wie geht man nun bei einer Textverarbeitung vor?

Zunächst erstellen Sie wie gewohnt den Brieftext. Überall da, wo später etwas eingefügt werden soll (etwa die Anschrift), setzen Sie einen speziellen "Platzhalter" ein, den die Textverarbeitung dann später gegen den tatsächlichen Text austauscht. Das könnte zum Beispiel so aussehen:

```
Abs.: ........

Herrn/Frau
#
#

#

Liebe# !

..... Brieftext ....
```

Das erste "#" steht in diesem Fall für den Vor- und Zunamen, das zweite für die Straße, das dritte für den Ort und das letzte für die Anrede.

Was Sie jetzt noch benötigen, sind die einzufügenden Texte. Im einfachsten Fall entnehmen Sie diese aus einer Dateiverwaltung, in der die Adressen der einzelnen Mitglieder gespeichert sind.

Sie können die Datenliste aber auch mit der Textverarbeitung erstellen. Dabei müssen Sie nur darauf achten, daß die einzelnen Daten voneinander getrennt sind, damit das Programm weiß, wo der eine Datensatz aufhört und der nächste beginnt. Für das Einladungsschreiben könnte das beispielsweise so aussehen:

```
Klaus Müller
Irgendwostr. 1
1000 Irgendingen 1
r Klaus
Else Maier
Irgendwostr. 2
2000 Irgendingen 2
Else
......
```

Diese Liste läßt sich natürlich beliebig fortschreiben. Sind alle Daten eingegeben, dann speichern Sie die Liste auf Diskette. Wenn Sie nun das Rundschreiben drucken möchten, müssen Sie der Textverarbeitung nur noch mitteilen, wo sie sich die einzufügenden Daten herholen soll. Die Serienbrief-Funktion holt sich jetzt den ersten Datensatz, fügt die Daten an den entsprechenden Stellen in den Text ein, druckt den Text, holt sich anschließend den zweiten Datensatz, usw...

Das eben geschilderte Verfahren stellt die einfachste Möglichkeit dar, einen Serienbrief zu erstellen. Die Serienbrief-Funktionen einiger Textverarbeitungen bieten aber noch wesentlich mehr. Ein gutes Beispiel dafür ist das in GEOS 2.0 enthaltene GEOMERGE in Verbindung mit GEOWRITE.

Anstelle eines einfachen Platzhalters kann man mit GEOMERGE auch sogenannte "Label" verwenden, die dann vor dem Drucken durch das Programm vom Anwender erfragt werden. Interessant ist das zum Beispiel, wenn Sie in einem Brief häufig den Nachnamen, etwa eines Kunden, verwenden wollen. In diesem Fall schreiben Sie an jede dieser Stellen ein Label "«Nachname»". GEOMERGE fragt Sie dann am Anfang einmal nach dem Nachnamen und fügt diesen dort ein, wo das entsprechende Label steht, egal, wie oft das Label im Text vorkommt.

Wer es noch flexibler haben möchte, dem steht eine spezielle IF-Anweisung für bedingte Texte zur Verfügung. Damit läßt sich in gewisser Weise sogar ein sogenanntes Textbausteinsystem einrichten, wie man es sonst nur im professionellen Bereich findet.

Besonders im geschäftlichen Bereich ist die Arbeit mit Textbausteinen sehr gebräuchlich, etwa um den laufenden Kontakt mit Kunden aufrechtzuerhalten. Wie sieht das nun konkret aus?

Insbesondere bei sehr großen Kundenstämmen wäre es natürlich viel zu aufwendig für jeden Kunden immer wieder ein individuelles Anschreiben zu verfassen. Daher verfügt der entsprechende Sachbearbeiter über eine größere Sammlung von Textbausteinen, von denen er sich jeweils die passenden heraussucht und dann zu einem Brief zusammensetzt. Solch ein Textbaustein ist in der Regel nicht mehr als ein oder zwei Absätze groß.

Aber auch das Heraussuchen und Zusammensetzen der passenden Bausteine ist "von Hand" noch sehr aufwendig, das kann man ebenfalls der Textverarbeitung überlassen. Nehmen wir dazu ein konkretes Beispiel. Eine Firma möchte an zwei Kunden einen Brief schreiben. (Diese beiden sollen jetzt einmal stellvertretend für vielleicht hunderte stehen.) Der eine Kunde hat gerade eine größere Bestellung getätigt, für die man sich bedanken will. Der andere Kunde hat schon lange nichts mehr bestellt und soll deshalb zu einer Bestellung ermuntert werden. Zwei mögliche Textbausteine könnten beispielsweise so aussehen:

```
Für Ihre Bestellung möchten wir uns recht herzlich
bedanken. Wir würden uns freuen, wenn Sie auch wei-
terhin bei uns bestellen würden.


Anbei erhalten Sie unseren neuesten Katalog. Sicher
werden auch Sie etwas darin finden, daß Ihnen
gefällt.
```

Um diese beiden Bausteine nun in einem "Formbrief" unterbringen zu können, definiert man bei GEOMERGE ein Label "guterKunde", das man wie folgt verwenden könnte:

```
«IF guterKunde="wahr"»
Für Ihre Bestellung möchten wir uns recht herzlich
bedanken. Wir würden uns freuen, wenn Sie auch wei-
terhin bei uns bestellen würden.«ENDIF»
```

```
«IF guterKunde="falsch"»
Anbei erhalten Sie unseren neuesten Katalog. Sicher
werden auch Sie etwas darin finden, daß Ihnen
gefällt.«ENDIF»
```

Falls es sich bei dem einzelnen Kunden also um einen guten Kunden handelt (guterKunde ist dann "wahr"), wird der erste Textbaustein gedruckt, ansonsten der Zweite!

Soviel zu den Serienbriefen und Textbausteinen. Gerade bei "offiziellen" Texten, wie etwa Einladungen oder Geschäftsbriefen, ist man natürlich besonders darauf bedacht, daß diese möglichst fehlerfrei geschrieben sind. Manche Leichtsinnsfehler entdeckt man aber auch nach dem zweiten oder dritten Durchlesen nicht. Da wäre es doch am besten, sich diese Arbeit ebenfalls vom Computer abnehmen zu lassen. Und das geht - zumindest in gewissem Umfang - in der Tat!

GEOS 2.0 beispielsweise verfügt über ein Rechtschreib-Prüfprogramm namens GEOSPELL, mit dem sich GEOWRITE-Texte überprüfen lassen.

Die Arbeitsweise eines solchen Programms ist in Grunde genommen ganz einfach: Das Programm verfügt über ein mehr oder minder umfangreiches Lexikon, in dem die korrekt geschriebenen Wörter abgelegt sind. Der zu überprüfende Text wird nun Wort für Wort durchgegangen. Jedes Wort wird dabei mit den Wörtern des Lexikons verglichen. Stimmt es mit keinem der Wörter im Lexikon überein, so gilt es als falsch geschrieben, und das Programm meldet sich mit einer entsprechenden Fehlermeldung. Das klingt jetzt nicht nur etwas umständlich, sondern ist es auch! Selbst bei professionellen Programmen im PC-Bereich ist die automatische Rechtschreibkontrolle meist ein sehr zeitraubendes Unterfangen.

Besonders wichtig sind der Umfang des Lexikons und die Möglichkeiten, das Lexikon um neue Begriffe zu erweitern. GEOSPELL beispielsweise bringt bei einem nicht erkannten Wort nicht einfach nur eine Fehlermeldung, sondern erlaubt es, das

Wort als neuen Begriff in das Lexikon aufzunehmen. Mit der Zeit wird das Lexikon dadurch immer umfangreicher und die Fehlermeldungen reduzieren sich in immer stärkerem Maße auf wirklich falsch geschriebene Wörter. Hat man diese Anfangsphase erst einmal überstanden, dann wird der Rechtschreibprüfer auch zu einer wirklichen Hilfe.

Besonders in der letzten Zeit hat ein Spezialbereich der Textverarbeitung zunehmende Bedeutung erlangt: die Einbindung von Grafik in den Text. Sieht man sich professionelle Druckerzeugnisse, wie etwa Zeitschriften, Werbebroschüren und ähnliches, an, so findet man Text und Bilder meistens bunt gemischt. Mit einer "normalen" Textverarbeitung lassen sich Grafiken nur sehr schwer verarbeiten. Zwar bietet beispielsweise auch GEOWRITE die Möglichkeit, Grafiken in den Text einzubinden. Das ist dann aber mit großen Einschränkungen verbunden, so daß es in der Regel allenfalls zu einem grafisch gestalteten Briefkopf reicht.

Besser ist es da, man bedient sich eines sogenannten Desktop-Publishing-Programms. Was man darunter genau versteht, das erfahren Sie im übernächsten Abschnitt. Sehen wir uns zunächst einmal an, wie man zu ansprechenden Grafiken gelangt.

2.2 Grafik

Neben der Textverarbeitung erfreut sich auch die Erstellung von Grafiken auf dem Commodore 64 sehr großer Beliebtheit. Hat man beim Malen oder Zeichnen eines Bildes doch noch am ehesten die Möglichkeit, sich kreativ auszutoben.

Um die grafischen Fähigkeiten des Commodore 64 nutzen zu können, benötigen Sie ein Mal- bzw. Zeichenprogramm. Über welche Eigenschaften und Unterschiede diese Programme verfügen, darüber möchte ich Ihnen im folgenden Auskunft geben.

Vielleicht haben Sie ja aber auch Lust, sich selbst mit der Grafikprogrammierung zu befassen. In diesem Fall empfiehlt sich die Lektüre von Kapitel 6. In diesem Kapitel erhalten Sie um-

fassende Informationen zu allen Aspekten der Grafikprogrammierung. Um die Grafik erfolgreich zu programmieren, sind allerdings gute Assembler-Kenntnisse erforderlich. Wenn Ihnen diese im Moment noch fehlen, sollten Sie zunächst einmal Kapitel 4 durcharbeiten.

Der Commodore 64 hat eine Grafikauflösung von 320*200 Punkten. Was meint man damit eigentlich konkret? Auf dem normalen Textbildschirm können Sie maximal 25 Zeilen zu je 40 Zeichen auf dem Bildschirm unterbringen, insgesamt also 2000 Zeichen. Man spricht hier von einer Auflösung von 25*40=2000 Zeichen. Bei der hochauflösenden Grafik dagegen haben Sie 64000 (320*200) Einzelpunkte, die jeder für sich gesetzt oder gelöscht werden können. Dadurch lassen sich sehr feine Grafiken erstellen. In dieser Auflösung kann man allerdings nur zwei Farben verwenden. Wer es bunter haben möchte, muß auf die niedrigere Auflösung von 160*200 Punkten zurückgreifen. Nicht alle Grafikprogramme unterstützen diese Auflösung. Dafür verfügen aber die meisten Programme über eine wesentlich größere "Zeichenfläche" als die eingangs erwähnten 320*200 Punkte. Erreicht wird dies durch die Aufteilung der Gesamtfläche in mehrere Einzelflächen, die getrennt bearbeitet werden, am Ende aber als eine große Fläche ausgedruckt werden können.

Wenn Sie auf herkömmliche Art und Weise ein Bild malen oder zeichnen wollen, nehmen Sie dazu einfach Papier und Bleistift oder Buntstifte. Wenn Sie mit dem Commdore 64 eine Grafik erstellen wollen, wird es schon etwas komplizierter. Sie können ja nicht einfach auf den Computerbildschirm "zeichnen". Irgendwie müssen Sie also die Grafik in den Rechner bekommen. Das einfachste Hilfsmittel dazu ist ein sogenannter Joystick, der ansonsten vor allem zum Spielen eingesetzt wird. Besser ist (insbesondere dann, wenn Sie auch "freihändig" zeichnen wollen) eine sogenannte Proportional-Maus.

Die meisten Grafikprogramme sind so aufgebaut, daß Sie mit dem Joystick oder der Maus einen Cursor - ähnlich dem gewohnten Textcursor - über den Bildschirm, die "Zeichenfläche", bewegen können. Drücken Sie dann zusätzlich auch noch die Maustaste oder den Feuerknopf des Joysticks, so werden an den

Stellen, über die der Cursor fährt, Punkte gesetzt. In diesem Fall spricht man dann vom "freihändig zeichnen". Natürlich ist das bei weitem nicht alles. Geometrische Grundformen, wie Rechtecke und Kreise etwa lassen sich mit den meisten Programmen durch spezielle Funktionen erstellen. Bei einem Rechteck zum Beispiel müssen Sie dann nur die linke obere und die rechte untere Ecke markieren. Den Rest erledigt das Zeichenprogramm.

Ein Mal- und Zeichenprogramm, über das jeder Commodore-64-Besitzer verfügt, ist GEOPAINT. Sehen wir uns dieses also einmal genauer an. Wie man GEOPAINT startet, müßten Sie noch aus Kapitel 1 wissen: einfach das GEOPAINT-Icon im Desktop doppelklicken.

Nachdem das Programm geladen wurde, erscheint auf dem Bildschirm eine sogenannte Dialogbox. Die drei Wahlmöglichkeiten erklären sich fast von selbst. Bitte wählen Sie die Option "Create" (bzw. "Erstellen", falls Sie bereits über GEOS 2.0 verfügen) zum Erstellen eines neuen Dokuments, und geben Sie anschließend einen beliebigen Namen an, beispielsweise "GRAFIK1". Sobald Sie die <Return>-Taste gedrückt haben, erscheint der GEOPAINT-Arbeitsbildschirm.

Die hell unterlegte Fläche in der Mitte stellt die Zeichenfläche dar. Dieses sogenannte Zeichenfenster zeigt allerdings immer nur einen kleinen Ausschnitt der Gesamtgrafik, der ausgedruckt etwa 8,25*4,5 cm groß wäre. Mit GEOPAINT lassen sich aber Abbildungen bis zum Format einer DIN-A-4-Seite erstellen.

In der rechten oberen Ecke sehen Sie den Titelbalken, der den Namen der Grafik enthält. Links daneben befindet sich die Menüleiste von GEOPAINT. Darunter sehen Sie die "Werkzeugleiste". Sie enthält die verschiedenen "Werkzeuge", die Ihnen GEOPAINT zur Bearbeitung Ihrer Grafik zur Verfügung stellt. Unterhalb der Werkzeugleiste befindet sich das "Musterquadrat". Manche Zeichenfunktionen, wie zum Beispiel die "Füllfunktion" zum Ausfüllen umrandeter Bereiche, erlauben die Verwendung verschiedener Füllmuster. Das Muster, das gerade eingestellt ist, wird dann im Musterquadrat dargestellt.

Rechts neben dem Musterquadrat ist der "Statusbereich". Er dient verschiedenen Zwecken. Im Augenblick sehen Sie ganz links den sogenannten "Seitenzeiger". Er markiert den Bereich der Gesamtgrafik, der gerade im Zeichenfenster dargestellt wird. Wenn Sie das kleine dunkle Rechteck anklicken und verschieben, wird ein anderer Ausschnitt der Grafik in das Zeichenfenster gebracht. Das kleine Quadrat neben dem Seitenzeiger zeigt die aktuelle Zeichenfarbe an. Die Leiste direkt daneben zeigt die 16 möglichen Zeichenfarben. Der Pfeil zeigt dabei auf die gerade verwendete Farbe.

Damit haben wir schon alle Bedienugselemente durch. Um nun ein bestimmtes Werkzeug (also eine bestimmte Zeichenfunktion) zu aktivieren, klicken Sie einfach dessen Symbol in der Werkzeugleiste an. Das Symbol wird daraufhin invertiert dargestellt, und die Zeichenfunktion steht zur Verfügung. Nach dem Start von GEOPAINT ist das Werkzeug "Bleistift" bereits aktiviert. Das erkennen Sie daran, daß sein Symbol in der vierten Zeile der Werkzeugleiste invertiert dargestellt ist.

Fahren Sie nun einmal mit dem Mauszeiger in das Zeichenfenster, und klicken Sie bzw. drücken Sie den Feuerknopf des Joysticks oder die linke Taste der Maus. Wenn Sie jetzt den Mauszeiger bewegen, werden in der Zeichenfläche fortlaufend Punkte gesetzt. Was Sie jetzt gerade machen, nennt man "freihändig zeichnen". Mit einem erneuten Klick können Sie die Zeichenfunktion abschalten, um zum Beispiel an eine andere Stelle der Zeichenfläche zu fahren. Mit dem Werkzeug "Bleistift" läßt sich schon einiges anfangen. Gerade Linien oder gar Rechtecke und Kreise kann man damit aber nur sehr schlecht zeichnen. GEOPAINT stellt dafür spezielle Werkzeuge zur Verfügung.

Wie zeichnet man z.B. eine gerade Linie? Klicken Sie dazu bitte das Gerade-Symbol in der Werkzeugleiste an. (Es befindet sich über dem Pinsel in der dritten Reihe rechts.) Fahren Sie jetzt wieder in das Zeichenfenster, und markieren Sie den Anfangspunkt der Geraden durch Klicken. Um exakt zeichnen zu können, zeigt Ihnen GEOPAINT im Statusbereich rechts unten die genauen Koordinaten auf der Zeichenfläche an. Nachdem Sie den Anfangspunkt markiert haben, fahren Sie mit dem Mauszei-

ger zu dem gewünschten Endpunkt der Geraden. GEOPAINT zeichnet nun permanent Linien zwischen dem Anfangspunkt und der aktuellen Position des Mauszeigers. Erst, sobald Sie ein zweites Mal geklickt haben, wird die Linie "fixiert".

Ähnlich gestaltet sich das Zeichnen von Rechtecken. Klicken Sie dazu das entsprechende Symbol in der Werkzeugleiste an (es befindet sich in der sechsten Zeile links). Anschließend markieren Sie durch Klicken die linke obere Ecke des Rechtecks und bewegen den Mauszeiger dann zur rechten unteren Ecke. Sobald Sie erneut klicken, wird das Rechteck endgültig gezeichnet.

Auch zum Zeichnen von Kreisen hat GEOPAINT ein spezielles Werkzeug. Sein Symbol finden Sie in der Werkzeugleiste unterhalb des Rechteck-Symbols. Nachdem Sie das Werkzeug aktiviert haben, markieren Sie zunächst den Mittelpunkt des Kreises durch Klicken. Anschließend fahren Sie an eine beliebige Stelle des Kreises. Sobald Sie ein zweites Mal klicken wird der Kreis fixiert.

Nicht selten kommt es vor, daß man sich verzeichnet. Arbeitet man mit Papier und Bleistift, so greift man dann zum Radiergummi. Auch in GEOPAINT steht Ihnen ein Radiergummi als Werkzeug zur Verfügung. Sein Symbol finden Sie in der Werkzeugleiste in der fünften Zeile links. Nachdem Sie das Symbol angeklickt haben, können Sie einen beliebigen Bereich der Grafik "ausradieren", d.h. löschen, indem Sie mit dem Mauszeiger darüberfahren.

Eine überaus nützliche Funktion, wenn es darum geht, einen Zeichenfehler auszubügeln, stellt die UNDO-Funktion dar. Haben Sie beispielsweise versehentlich einen Kreis an der falschen Stelle gezeichnet, so dürfte es Ihnen äußerst schwer fallen, diesen mit dem Radiergummi wieder aus der Grafik zu entfernen, ohne nicht auch andere Teile der Grafik zu zerstören. In diesem Fall genügt ein Anklicken des UNDO-Symbols ganz unten auf der Werkzeugleiste. UNDO macht immer die jeweils letzte Zeichenoperation rückgängig, egal, ob es sich dabei um einen Kreis, ein Rechteck oder sonst eine Zeichenfunktion handelt.

Einen etwas ungewöhnlichen Weg geht GEOPAINT bei der Farbgestaltung. Wie ich schon eingangs erwähnt habe, gibt es auf dem Commodore 64 grundsätzlich zwei Darstellungsmodi für die hochauflösende Grafik. Im einen Modus kann man bis zu vier Farben verwenden. Dafür ist die Auflösung mit nur 160*200 Punkten aber so niedrig, daß man schon fast nicht mehr von hochauflösender Grafik sprechen kann. GEOPAINT arbeitet mit der Auflösung von 320*200 Punkten, in der an sich nur zwei Farben verwendet werden können. Durch einen Trick kann man aber mit allen 16 Farben arbeiten.

Dieser Trick besteht darin, daß man die Hintergrund- und Punktfarbe immer nur für einen 8*8 Punkte großen Bereich festlegen kann. Warum das so ist, werden Sie in Kapitel 6 erfahren, wenn wir uns selbst der Grafikprogrammierung widmen werden.

Zum Ändern der Farben klicken Sie das COLOR-Symbol in der Werkzeugleiste an. Im Statusbereich rechts unten auf dem Bildschirm erscheinen nun zwei Farbleisten. In der oberen stellen Sie die Farbe der gesetzten Punkte ein, die untere dient zum Einstellen der Hintergrundfarbe.

Neben den gerade vorgestellten bietet GEOPAINT natürlich noch wesentlich mehr Funktionen, deren Bedeutung Sie einfach durch Ausprobieren herausfinden können. Im Zweifelsfall hilft ein Blick ins Handbuch.

GEOPAINT ist mit Sicherheit ein ausgezeichnetes Zeichenprogramm mit gut durchdachten Funktionen. Über den größten Nachteil von GEOPAINT werden Sie sich sicher aber auch schon geärgert haben: das ständige Nachladen von Disketten.

Eine interessante und mit 58 Mark auch sehr preiswerte Alternative zu GEOPAINT stellt das Programm "Eddison" von Scanntronik dar. Auch Eddison arbeitet am besten mit einer Proportional-Maus zusammen und läßt sich fast vollständig über Menüs steuern.

Eddison arbeitet mit einer maximalen Zeichenfläche von 640*400 Punkten. Das entspricht vier normalen Bildschirmen. Jeder dieser Bildschirme läßt sich auch für sich allein nutzen.

Die Zeichenfunktionen von Eddison lassen kaum Wünsche offen. Man kann freihändig zeichnen, Linien ziehen, Rechtecke, Kreise und Ellipsen zeichnen, und natürlich steht auch eine Füllfunktion mit verschiedensten Füllmustern zur Verfügung.

Alles läuft dabei mit größtmöglichem Bedienungskomfort ab. Rechtecke und Kreise beispielsweise kann man ähnlich wie bei GEOPAINT mit einem "Gummiband" zeichnen, die Undo-Funktion macht mißlungene Zeichenversuche problemlos rückgängig.

Sehr nützlich ist auch die Übersichtsfunktion. Klickt man dazu das entsprechende Icon an, zeigt Eddison eine auf die Hälfte verkleinerte Gesamtdarstellung des Grafikspeichers.

Der wohl umfangreichste und leistungsfähigste Befehl ist der Move-Befehl. Was sich damit alles anstellen läßt, ist schon erstaunlich. Move beschränkt sich nämlich nicht nur auf das Verschieben von Grafikbereichen, wie der Name vermuten lassen könnte. Move ist zusätzlich in der Lage, beliebige rechteckige Bildschirmbereiche zu spiegeln und sogar zu vergrößern oder zu verkleinern. Daß man Bildschirmbereiche mittels UND, ODER und EXCLUSIV-ODER logisch miteinander verknüpfen kann, versteht sich schon fast von selbst.

Das Malen von Bildern ist zwar ganz schön, oft möchte man aber exakte Zeichnungen erstellen, zum Beispiel Schaltpläne. Der Fachmann spricht dann übrigens von CAD, was für "Computer Aided Design" steht.

Eddison bietet zwei Spezialitäten, die ihn schon fast zu einem kleinen CAD-Programm machen: eine "permanente Koordinatenanzeige" und sogenannte "Construction Sets". Da wäre zunächst die Koordinatenanzeige. Diese informiert einen ständig über die aktuelle Position des Grafik-Cursors. Entweder in Form von X/Y-Koordinatenangaben oder in Form von Millimeterangaben. Mit Hilfe der Funktionstasten F7 und F8 lassen sich auch

Tabulatorpunkte markieren, die dann durch einen Druck auf die Funktionstaste F7 direkt angesprungen werden. Hilfreich ist das zum Beispiel, wenn man von einer Stelle aus mehrere Linien ziehen möchte. Ebenfalls sehr nützlich ist ein sogenanntes Punktraster, das man über seine Zeichnung legen lassen kann.

Bei technischen Zeichnungen hat man in der Regel einiges an gleich aussehenden Teilen: bei einem elektrischen Schaltplan zum Beispiel Widerstände, Kondensatoren, Transistoren usw... Da ist es natürlich am einfachsten, wenn man diese Teile jeweils nur einmal zeichnen muß und dann immer wieder verwenden kann.

Eddison bietet dazu einige interessante Sprite-Befehle, mit Hilfe derer sich ein Grafikbaukasten, ein Construction Set, realisieren läßt. Ein "Sprite" ist ein kleines Grafikobjekt von 24*21 Punkten Größe, das sich frei über den Bildschirm bewegen läßt. Der Grafik-Cursor zum Beispiel besteht aus solch einem Sprite.

Mit Hilfe des Sprite-Befehls "Get" ist es nun möglich, einen beliebigen 24*21 Punkte großen Grafikausschnitt in ein Sprite zu kopieren. Der Grafik-Cursor hat jetzt das Aussehen des Grafikausschnitts. Das Sprite kann nun beliebig oft wieder in die Grafik eingefügt werden. Entweder man "stempelt" es in die Grafik (dabei wird dann der darunterliegende Grafikbereich zuvor nicht gelöscht) oder man "klebt" es wie eine Briefmarke in die Grafik, wobei dann der Grafikbereich zuvor gelöscht wird. Mit Hilfe eines speziellen Sprite-Editors läßt sich der Grafikausschnitt auch beliebig verändern.

Mit Hilfe dieser Technik läßt sich der Entwurf einer technischen Zeichnung drastisch vereinfachen. Nehmen wir dazu zum Beispiel einmal an, Sie wollen häufiger Schaltpläne entwerfen. Dazu benötigt man natürlich vor allem die verschiedenen Schaltsymbole, etwa für Widerstände und Transistoren.

Zunächst müßten Sie also daran gehen, diese Symbole als 24*21 Punkte große Grafiken (damit sie in ein Sprite passen) zu entwerfen. Am einfachsten geht das mit dem schon erwähnten Sprite-Editor. Sobald ein Symbol fertig ist, kopieren Sie es in einen freien Grafikbildschirm. Nach und nach kommen so viel-

leicht 20 bis 30 einzelne Symbole zusammen, die alle zusammen auf einem Grafikbildschirm untergebracht werden sollten. Am Ende speichern Sie diesen Bildschirm auf Diskette ab, und schon haben Sie Ihren Schaltplan-Baukasten!

Wenn Sie jetzt eine Schaltung entwerfen wollen, laden Sie zuerst die Grafik mit den Symbolen in einen freien Bildschirm. Auf einem anderen freien Bildschirm zeichnen Sie dann die Schaltung. Benötigen Sie ein bestimmtes Schaltsymbol, so gehen Sie schnell in die Schaltsymbol-Grafik, holen sich das betreffende Symbol in das Sprite, schalten anschließend wieder auf ihre Schaltung um und kopieren das Symbol an der vorgesehenen Stelle in die Grafik. (Sollten zuvor noch Detail-Änderungen erforderlich sein, so gehen Sie schnell in den Sprite-Editor und führen diese dort durch.) Eine an sich sehr einfache, aber ungeheuer effiziente Technik!

Zum Schluß möchte ich noch kurz auf ein Programm eingehen, das man als den "großen Bruder des Eddison" bezeichnen kann: Eddifox. Eddifox bietet Grafikfunktionen, die man selbst bei manchen Grafikprogrammen aus dem professionellen Bereich vergebens sucht. Das hat allerdings auch seinen Preis. Eddifox selbst ist zwar mit 88 Mark nicht gerade teuer. Eddifox läuft dafür aber nur zusammen mit dem Pagefox-Modul, das ich Ihnen im nächsten Unterkapitel vorstelle, und das kostet immerhin 248 Mark! Eddifox bietet einen Grafikspeicher von 640*800 Punkten, was einer ganzen DIN-A4-Seite oder 8 normalen Bildschirmen entspricht. Alle Zeichenfunktionen sind dabei nicht wie bei anderen Programmen auf den jeweils sichtbaren Teil einer Grafikseite beschränkt, sondern können im Extremfall auf die ganze Seite wirken.

Die herausragendste Fähigkeit des Programms stellt aber die Möglichkeit dar, beliebige Grafikausschnitte zu verzerren. Das hört sich zunächst vielleicht unscheinbar an, birgt aber gigantische Möglichkeiten. Ein vorhandener rechteckiger Grafikbereich läßt sich durch die Verzerrung in eine beliebige andere Form bringen, ja sogar auf einen dreidimenionalen Körper, etwa einen Zylinder, "wickeln".

Wer mit dem Commodore 64 professionell aussehende Grafiken erstellen möchte, vielleicht um diese fürs Desktop Publishing zu verwenden, wird am Eddifox kaum vorbeikommen. Doch "Desktop Publishing", was ist das eigentlich?

2.3 Desktop Publishing

Unter Desktop Publishing (kurz: DTP) versteht man die Gestaltung von Drucksachen aller Art. Das können sowohl einfache Werbezettel oder Geburtstagseinladungen, im Extremfall aber auch komplette Zeitungen oder Zeitschriften sein.

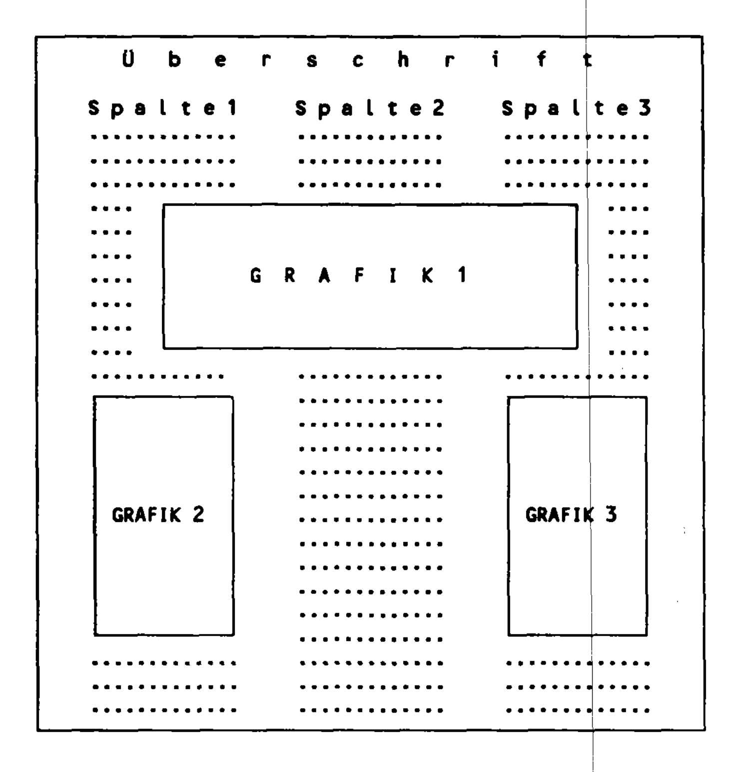
Das Entscheidende dabei: Im Gegensatz zu einem "normalen" Textverarbeitungsprogramm ist man mit einem Desktop-Publishing-Programm in der Lage, Text und Grafik gemeinsam zu verarbeiten.

Mit Hilfe eines Desktop-Publishing-Programms wird man also sozusagen zum "Schreibtisch-Verleger". Während im professionellen Bereich der Druckgestaltung bisher sehr viele Einzelschritte erforderlich waren, bis eine bestimmte Drucksache "druckreif" bzw. tatsächlich gedruckt war, lassen sich beim Desktop Publishing alle Schritte am Schreibtisch bzw. auf dem Computerbildschirm erledigen:

Zunächst werden der Text und die Grafiken separat erstellt und anschließend das ganze am Bildschirm seitenweise zusammenmontiert. Der Fachmann spricht hier vom "Layouten" der Drucksache. Anschließend wird die Drucksache über einen hochwertigen Drucker in beliebiger Stückzahl ausgedruckt.

Das hört sich nun alles sehr einfach an, führt aber in der Praxis selbst bei professionellen Systemen zu zum Teil erheblichen Problemen. Insbesondere das Einbinden der Grafiken in den Text bereitet in der Regel erhebliches Kopfzerbrechen. Noch mehr als schon die Textverarbeitung steht und fällt das Desktop Publishing aber mit dem zur Verfügung stehenden Drucker. Zeitungen und Zeitschriften werden heutzutage in der Regel mit sogenannten Fotosatzbelichtern "gedruckt", die sich in Preisregionen

jenseits von 50000 Mark bewegen. Eine Stufe tiefer stehen die sogenannten Laserdrucker, die in letzter Zeit zwar erheblich billiger geworden sind, aber immer noch über 5000 Mark kosten.



Beide Druckertypen sind für den Einsatz am Commodore 64 natürlich völlig indiskutabel. Mit einem einfachen Matrixdrucker lassen sich aber ebenfalls durchaus akzeptable Druckergebnisse erzielen, vorausgesetzt, man hat das richtige DTP-Programm.

Wie sieht nun DTP auf dem Commodore 64 aus? Dazu wollen wir uns einmal zwei sehr gelungene Produkte von Scanntronik anschauen: den Pagefox und den Printfox.

Der Pagefox wird auf einem Steckmodul geliefert, auf dem sich zusätzlich 100 Kbyte Speicher befinden. Mit 248 Mark kann man den Pagefox zwar nicht gerade als billig bezeichnen, er ist

aber auf jeden Fall sein Geld wert. Der Printfox ist sozusagen der kleine Bruder des Pagefox und für den kleineren Geldbeutel gedacht. Er kostet nur 98 Mark und wird dafür aber auch nur auf Diskette geliefert, verfügt also über keinen zusätzlichen Speicher.

Wer nur gelegentlich Drucksachen erstellen möchte, etwa Geburtstagseinladungen oder ähnliches, ist mit dem Printfox sicher gut bedient. Der Printfox besteht im wesentlichen aus zwei Teilen: einem Texteditor vergleichbar einer Textverarbeitung und einem Grafikeditor für die hochauflösende Grafik.

Nach dem Start des Programms befindet man sich im Texteditor. Dieser verfügt über alle Funktionen einer guten Textverarbeitung. So kann man mit Hilfe der Cursor-Tasten frei im Text herumwandern. Wörter, die nicht mehr ganz in eine Zeile passen, werden ganz in die nächste Zeile geholt (Word-Wrapping), und es gibt Blockoperationen zum Einfügen, Löschen und Überschreiben von Textteilen. Und natürlich fehlen auch die Funktionen zum Suchen und Ersetzen von Textstücken nicht.

Der Textspeicher ist auf 8030 Zeichen begrenzt. Im Vergleich zu reinen Textverarbeitungen ist das zwar nicht allzu viel, reicht aber in der Regel vollkommen aus. Man möchte ja meist nur ein einseitiges oder allenfalls ein zwei- bis dreiseitiges Dokument erstellen.

Seine eigentliche Stärke offenbart der Texteditor, wenn es ans Formatieren geht. Der Printfox verfügt nämlich über eine richtige kleine Programmiersprache! Das hat zwar den Nachteil, daß man sich mit dieser erst einmal vertraut machen muß. Die ungeheure Flexibilität, mit der sich die Texte dadurch formatieren lassen, macht dies aber schnell wieder wett.

Die Formatierungsanweisungen werden in Formatzeilen abgelegt, die im Prinzip an beliebiger Stelle im Text stehen können. Eine Formatzeile könnte beispielsweise so aussehen:

```
z=1 s=1 g=0 h=1 v=2 x=30 y=6 l=280 t=20,50 i=780 x=330 y=6
```

Was einem sofort auffällt ist die einheitliche, GEOS-ähnliche Benutzeroberfläche. Wer also mit GEOS vertraut ist, wird mit dem Pagefox sofort zurechtkommen. Fast alle Programmfunktionen lassen sich wahlweise aber auch über die Tastatur aufrufen. Der Pagefox besteht aus drei Programmteilen:

- Dem Texteditor zur Eingabe und Bearbeitung des Textes.
- Dem Grafikeditor zum Erstellen und Bearbeiten von Grafiken.
- Dem Layout-Editor zur Anordnung von Text und Grafik auf einer Seite.

Alle drei Programmteile sind im Steckmodul untergebracht und stehen daher jeweils blitzschnell zur Verfügung.

Der Grafik- und der Texteditor haben beim Pagefox dieselbe Funktion wie beim Printfox. Dank der eingebauten Speichererweiterung läßt sich im Grafikeditor aber eine komplette DIN-A4-Seite bearbeiten. Das entspricht einer Auflösung von 640*800 Punkten oder 8 Bildschirmen. Damit man nicht die Übersicht verliert, kann man sich die Grafik auf 50 oder 25 Prozent verkleinern lassen.

Der Grafikeditor verfügt über zahlreiche Funktionen, die alle über zwei Menüleisten erreichbar sind. Eine kleine Übersicht soll hier genügen:

- Koordinatenanzeige
- Texteditor
- Layout-Editor
- Verkleinerung 50%
- Zoom/Sprite-Editor
- Erase
- Stamp
- Append
- Get
- Text-Funktion
- Move
- Spray-Funktion
- Flächen füllen
- Kreise und Ellipsen
- Move 1 Punkt
- Move 8 Punkte
- Scrollen/Verschieben
- Löschen
- Linien
- dickerer Pinsel
- Freihändig zeichnen
- Undo
- Drucken
- Disk-Befehle
- Speichern
- Laden
- Invertieren
- Punktgitter
- Und-Verknüpfung
- Exor-Verknüpfung
- Oder-Verknüpfung
- Rechtecke

Anstelle des vorhandenen Grafikeditors kann man auch das Programm Eddifox verwenden, das im vorigen Unterkapitel kurz vorgestellt wurde. Eine Besonderheit des Texteditors ist die automatische Silbentrennung, d.h., Wörter, die nicht mehr ganz in eine Zeile passen, werden automatisch grammatikalisch richtig getrennt. Allerdings funktioniert dies nicht immer, da die deutsche Grammatik ja bekanntlich reich an Ausnahmen ist. Die "Trefferquote" des Trennalgorithmus soll aber bei über 90% liegen.

Ungewöhnlich vielfältig sind die Schriftvariationen:

- Tiefstellen
- Hochstellen
- Shadow
- Outline
- Kursiv
- Unterstreichen
- Doppelt hoch
- Doppelt breit
- Fettdruck

Fast alle Variationen können auch miteinander kombiniert werden. Dadurch läßt sich das Schriftbild sehr individuell gestalten.Der Text- und der Grafikeditor sind schon für sich allein sehr mächtige Werkzeuge. Das Herzstück des Pagefox bildet aber der Layout-Editor. Mit ihm kann man die separat mit dem Text- bzw. Grafikeditor erstellten Texte und Grafiken auf einer Seite zusammenmontieren.

Um die Arbeitsweise des Layout-Editors besser zu verstehen, sollte man wissen, wie ein professioneller Layouter arbeitet. Nehmen wir beispielsweise eine Zeitschriftenseite. Diese besteht in der Regel aus einer mehr oder minder bunten Mischung aus Texten und Abbildungen oder Fotos.

Trotz ansonsten modernster Technik sind die beiden wichtigsten Werkzeuge eines Layouters "Schere und Klebstoff". Die verschiedenen Teile einer Seite werden mit der Schere passend zurechtgeschnitten und dann auf einer Unterlage aufgeklebt.

Anstelle von Schere und Klebstoff hat man beim Layout-Editor des Pagefox sogenannte "Rahmen", die zwar grundsätzlich rechteckig sind, sich dafür aber jederzeit vergrößern, verkleinern oder auch verschieben lassen.

Die zu erstellende Seite wird also in einzelne Rahmen beliebiger Größe unterteilt. In einem Rahmen kann beispielsweise die Überschrift stehen, in einem anderen Rahmen eine Grafik und in den restlichen Rahmen der normale Text in verschiedenen Schriftgrößen.

Wie montiert man nun eine Seite konkret? Gehen wir dazu einmal die einzelnen Schritte durch: Zunächst werden die Textrahmen festgelegt. Dazu wird die betreffende Funktion durch Anklicken ihres Icons aktiviert. Anschließend kann man einen einzelnen Rahmen genauso festlegen wie ein einfaches Rechteck im Grafikeditor, d.h., man markiert durch zwei Mausklicks zwei diagonale Ecken des Rechtecks.

Die Grafikrahmen werden analog festgelegt. Nur muß man hier gleich die zugehörige Grafik angeben, die aber dann erst später beim Formatieren in die Seite eingefügt wird.

Nun hat man also die einzelnen Rahmen eingezeichnet. In der Regel wird aber jetzt noch nicht alles so hinhauen, wie man sich das ursprünglich vorgestellt hat. Hier zeigt der Editor seine wahre Stärke. Jeder Rahmen, egal ob Text- oder Grafikrahmen, kann nachträglich beliebig auf der Seite verschoben und sogar vergrößert oder verkleinert werden. Möglich wird dies, da die Rahmen als einzelne "Objekte" aufgefaßt werden, d.h., sie werden nicht nur einfach in die hochauflösende Grafik gezeichnet, sondern ihre Koordinaten und ihr Aussehen werden zusätzlich separat gespeichert, so daß sie sich an ihrer ursprünglichen Position leicht löschen und woanders plazieren lassen.

Wie soll nun der Text in den einzelnen Rahmen erscheinen? Dazu muß man die Satzart festlegen. Zur Auswahl stehen: linksbündig, Blocksatz, zentriert, rechtsbündig, glatt an Grafik, außen glatt sowie global zentriert. Besonders interessant ist die

Möglichkeit, Text um eine Grafik "herumfließen" zu lassen. Dabei wird der Text gerade so um die Grafik herumgeschrieben, daß er diese nicht berührt.

Hat man alle Seitenattribute festgelegt, gilt es, sich einen ersten Überblick über das spätere Aussehen der Drucksache zu machen. Dazu gibt es die Schnellformatierung, die nur wenige Sekunden in Anspruch nimmt. Bei der Schnellformatierung wird nur der Zeilen- und Spaltenumbruch berechnet und die Zeilen werden als Linien dargestellt. Außerdem erhält man, falls der vorhandene Text nicht ganz in die Rahmen paßt, eine entsprechende Fehlermeldung.

Den Abschluß des Layoutens bildet die Vollformatierung. Anschließend steht die komplette Seite druckfertig im Grafikspeicher, wo sie sich mit Hilfe des Grafikeditors gegebenenfalls auch noch verändern läßt.

Da eine ganze DIN-A4-Seite ja nicht komplett auf den Bildschirm paßt, hat man die Möglichkeit, sich die Seite in verkleinerter Form am Bildschirm zeigen zu lassen. Dabei ist dann zwar der Text nicht lesbar, dafür sieht die Seite aber exakt so aus wie auch später auf dem Papier. Für eine letzte Kontrolle vor dem Ausdruck ist das sehr nützlich, da das Ausdrucken ja - je nach Drucker - erhebliche Zeit in Anspruch nimmt.

Natürlich kann man mit dem Pagefox auch mehrseitige Dokumente erstellen. Die einzelnen Seiten werden dann einfach der Reihe nach von Diskette nachgeladen.

Mit seinen 248 Mark ist der Pagefox nicht ganz billig. Deshalb stellt sich natürlich die Frage, ob es sich überhaupt lohnt, mit einem Heimcomputer wie dem Commodore 64 ins Desktop Publishing einzusteigen.

Rechnet man alles zusammen (Commodore 64, Floppy, Drucker und Pagefox-Modul), so kommt man auf Anschaffungskosten von etwa 2000 Mark. Für diesen Preis bekommt man im profes-

sionellen DTP-Bereich noch nicht einmal ein DTP-Programm. (Wirklich gute DTP-Programme im PC-Bereich kosten über 3000 Mark.)

Professionelle DTP-Programme können natürlich einiges mehr als der Pagefox. Ob man diese Fähigkeiten dann nachher auch benötigt, ist aber eine ganz andere Frage. Und, daß man mit dem Pagefox vernünftig arbeiten kann, zeigt nicht zuletzt sein Handbuch. Es umfaßt immerhin rund 70 Seiten und wurde komplett mit dem Pagefox erstellt!

Um nur gelegentlich beispielsweise eine Geburtstagseinladung anzufertigen, sind 2000 Mark sicher zuviel. Wer aber regelmäßig kleinere Drucksachen erstellen muß, etwa Werbezettel, Preislisten oder Vereinszeitungen, für den dürfte DTP mit dem Commodore 64 und dem Pagefox eine ernstzunehmende Alternative sein.

2.4 Datenverwaltung

Neben der Textverarbeitung nimmt auch die Datenverwaltung einen sehr breiten Raum im Bereich der Computeranwendungen ein. Das Hauptargument dabei ist die Schnelligkeit und Flexibilität mit der ein Computer bestimmte Informationen finden kann. Auch der Commodore 64 eignet sich sehr gut zur Datenverwaltung, vorausgesetzt, die Datenmengen sind nicht allzu groß. Um nun Ihre Daten mit dem 64'er verwalten zu können, benötigen Sie ein passendes Programm. Das Angebot ist dabei ähnlich vielfältig wie bei den Textverarbeitungen. Grundsätzlich kann man zwei Arten von Datenverwaltungen unterscheiden: Dateiverwaltungen und Datenbanken.

Die meisten Datenverwaltungsprogramme sind Dateiverwaltungen. Die einzige wirkliche Datenbank für den Commodore 64 ist das Programm "Superbase", das sich durchaus auch mit Programmen aus dem professionellen Bereich messen kann. Worin liegen nun die grundsätzlichen Unterschiede zwischen einer Dateiverwaltung und einer Datenbank?

Bei einer Dateiverwaltung werden Ihre Daten zu einzelnen Dateien zusammengefaßt. Das kann beispielsweise eine Adreßdatei sein, eine Telefonverzeichnis-Datei oder eine Schallplattendatei. Jede Datei besteht aus einzelnen Datensätzen, diese wiederum aus einzelnen Datenfeldern. Bei einer Adreßdatei könnte ein Datensatz zum Beispiel aus dem Vor- und Zunamen, der Straße sowie dem Ort bestehen. Eine Dateiverwaltung läßt sich noch am ehesten mit einem Karteikasten vergleichen, die einzelnen Datensätze stellen dann die Karteikarten dar.

Was die Eingabe der Daten betrifft, sind sich Dateiverwaltung und Karteikasten recht ähnlich. Ihre eigentliche Stärke offenbart die Dateiverwaltung bei der Datensuche. Einen Karteikasten können Sie ja im Grunde genommen nur von vorne bis hinten durchblättern, bis Sie die gesuchte Karteikarte gefunden haben. Mit einer Dateiverwaltung läßt sich da wesentlich mehr machen!

Auch bei einer Datenbank sind die Daten in einzelnen Dateien zusammengefaßt. Im Gegensatz zu einer Dateiverwaltung ist eine Datenbank aber in der Lage, zwischen den einzelnen Dateien Querverbindungen herzustellen, d.h., einzelne Daten einer Datei können Bestandteil einer anderen Datei sein. Außerdem verfügt eine Datenbank über eine eigene Programmiersprache, mit der man komplette Anwendungsprogramme schreiben kann, die die in der Datenbank gespeicherten Daten nutzen.

Wie gesagt ist Superbase das einzige Programm auf dem Commodore 64, das den Namen Datenbank verdient. Mit nur 99 Mark ist es zudem recht günstig zu haben. Auch die meisten Dateiverwaltungen, wie etwa Datamat, das ebenfalls 99 Mark kostet, bewegen sich in dieser Preisregion. Da stellt sich natürlich die Frage, ob man sich anstelle einer einfachen Dateiverwaltung nicht lieber gleich Superbase kaufen soll.

Nun, das hängt vor allem davon ab, welche Art von Daten Sie verwalten wollen. Entscheidend ist aber auch, wie intensiv Sie sich mit der ganzen Materie befassen wollen.

Um die Fähigkeiten eines Programms, wie Superbase, voll nutzen zu können, muß man sich mit dem Programm sehr eingehend beschäftigen, was natürlich eine Menge Zeit kostet.

Mit einer Dateiverwaltung, wie Datamat, können Sie dagegen sofort loslegen, ohne sich groß mit der Handhabung und Bedienung des Programms herumzuschlagen. Die meisten Programmfunktionen erschließen sich Ihnen so ganz nebenbei während der Arbeit mit dem Programm.

Nehmen wir als Beispiel einmal eine Adreßdatei. Zunächst müssen Sie festlegen, welche Daten Sie konkret speichern wollen. Ein Datensatz wird dazu in einzelne Datenfelder unterteilt. Das könnte beispielsweise so aussehen:

```
Nachname: <                   >
Vorname:  <                   >

Strasse:  <                   >
Wohnort:  <                   >
PLZ:      <    >

Telefon:  <             >

Bemerkung:<                                  >
```

Zwischen den "<" und ">" können Sie später die einzelnen Daten eingeben. Die meisten Programme sind bei der Gestaltung der Eingabemaske sehr flexibel. So kann man die Anzahl der Datenfelder und die Länge der einzelnen Datenfelder in der Regel frei wählen (natürlich innerhalb gewisser Grenzen).

Haben Sie die Eingabemaske definiert, kann es mit der Dateneingabe losgehen. Eine Beispielseingabe in die obige Maske könnte zum Beispiel so aussehen:

```
Nachname: <Müller             >
Vorname:  <Egon               >

Strasse:  <Irgendwostr. 2     >
Wohnort:  <Irgendingen        >
```

```
PLZ:        <1234>

Telefon:    <12 34/56 78 90>

Bemerkung:<Hat am 15.12. Geburtstag                    >
```

Durch einen Aufruf der entsprechenden Funktion wird der eingegebene Datensatz auf der Diskette dauerhaft abgespeichert und die Eingabemaske für die nächste Eingabe gelöscht. Je nach Größe der Datensätze lassen sich in einer einzelnen Datei bis zu mehrere hundert Datensätze unterbringen.

Nehmen wir nun einmal an, Sie haben in die Adreßdatei eine größere Anzahl von Daten eingegeben und möchten jetzt eine bestimmte Adresse heraussuchen. Welche Möglichkeiten stellt Ihnen eine gute Dateiverwaltung dazu zur Verfügung?

Praktisch jede Dateiverwaltung arbeitet zumindest mit einem sogenannten "Indexfeld", das man beim Einrichten der Datei festlegen muß. In unserer Adreßdatei könnte das beispielsweise das Nachname-Feld sein. Sämtliche Inhalte des Indexfeldes werden komplett im Rechnerspeicher gehalten und stehen daher blitzschnell zur Verfügung.

Wenn Sie jetzt zu einem bestimmten Nachnamen die zugehörige Adresse suchen, vergleicht die Dateiverwaltung den Nachnamen mit den gespeicherten Indizes (da diese meist alphabetisch sortiert sind, geht das sehr schnell) und holt dann den entsprechenden Datensatz von der Diskette.

Eine weitere Möglichkeit, an bestimmte Daten zu kommen, ist das "Durchblättern" der Datei wie bei einem Karteikasten. Die Dateiverwaltung verfügt dazu über Funktionen, mit denen man sich den jeweils nächsten oder den vorherigen Datensatz (bezogen auf den Datensatz, der sich gerade am Bildschirm befindet) oder auch den ersten und den letzten Datensatz einer Datei zeigen lassen kann. Diese Methode eignet sich natürlich nur bei einem relativ kleinen Datenbestand, sonst hätte man ja gleich beim Karteikasten bleiben können und sich nicht extra eine Dateiverwaltung kaufen müssen.

Viel interessanter ist da die Suche nach bestimmten Suchkriterien. Hier zeigt eine Dateiverwaltung ihre wahre Stärke. Oft kommt es ja vor, daß man von einer gesuchten Adresse nur ein bestimmtes Datenfeld kennt, beispielsweise den Vornamen oder die Straße. Eine "Anfrage" an die Dateiverwaltung könnte dann so aussehen:

```
Nachname: <*                   >
Vorname:  <Egon                >

Strasse:  <*                   >
Wohnort:  <*                   >
PLZ:      <*   >

Telefon:  <*              >

Bemerkung:<*                                    >
```

oder so:

```
Nachname: <*                   >
Vorname:  <*                   >

Strasse:  <Irgendwostr.        >
Wohnort:  <*                   >
PLZ:      <*   >

Telefon:  <*              >

Bemerkung:<*                                    >
```

Die Dateiverwaltung sucht daraufhin alle Personen heraus, die mit Vornamen Egon heißen bzw. in der Irgendwostraße wohnen. Die Sterne (*) in den anderen Datenfeldern signalisieren der Dateiverwaltung, daß diese Felder bei der Suche nicht berücksichtigt werden sollen.

Natürlich lassen sich auch mehrere Suchkriterien miteinander verknüpfen:

```
Nachname: <*                    >
Vorname:  <Egon                 >

Strasse:  <*                    >
Wohnort:  <Irgendingen          >
PLZ:      <*   >

Telefon:  <*              >

Bemerkung:<*                                        >
```

Die Dateiverwaltung sucht nun alle Egons, die in Irgendingen wohnen.

Diese Abfragen sind schon recht komplex (verglichen mit den Möglichkeiten eines Karteikastens). Es geht aber noch raffinierter. Dazu verfügen die meisten Dateiverwaltungen über ein sogenanntes "Jokerzeichen". Ein Jokerzeichen (zum Beispiel das Fragezeichen "?") kann in einer Abfrage für jedes beliebige andere Zeichen stehen:

```
Nachname: <M??er                >
Vorname:  <*                    >

Strasse:  <*                    >
Wohnort:  <Irgendingen          >
PLZ:      <*   >

Telefon:  <*              >

Bemerkung:<*                                        >
```

Durch diese Angabe werden alle Personen herausgesucht, die in Irgendingen wohnen und "Mayer", "Maier" oder "Meier" heißen! Der zweite und der dritte Buchstabe des Nachnamens sind in diesem Fall bei der Suche ohne Belang.

Hat man die gesuchten Daten erst einmal gefunden, möchte man natürlich etwas mehr damit anfangen können, als sie nur am Bildschirm lesen zu können. Zwar hat man bei einer Dateiverwaltung nicht die Weiterverarbeitungsmöglichkeiten wie bei ei-

ner Datenbank. Zwei Grundfunktionen stellt aber fast jede Dateiverwaltung zur Verfügung: das Ablegen der Daten in einer separaten Datei und das Ausdrucken der Daten.

Nachdem die Daten in einer separaten Datei abgelegt wurden, können Sie sie mit einem anderen Programm weiterverarbeiten, beispielsweise einer Textverarbeitung, um Serienbriefe zu erstellen.

Zum Ausdrucken der Daten erlaubt beispielsweise Datamat die Erstellung einer speziellen Druckmaske (ähnlich der Eingabemaske), mit der man die Daten leicht in das gewünschte Druckformat bringen kann. Adressen etwa möchte man ja vielleicht auf Adreßaufklebern aufdrucken, wobei in diesem Fall alles außer dem Vor- und Nachnamen, der Straße und dem Wohnort überflüssig ist. In unserer Adreßdatei dürften also die Telefonnummer sowie die Bemerkungen nicht gedruckt werden. Mit der Druckmaske kein Problem.

Neben der Datensuche stellt auch die Pflege der Daten einen ganz wichtigen Punkt dar. Damit meint man die Veränderung vorhandener Datensätze einer Datei. Es könnte ja zum Beispiel sein, daß ein Freund umzieht oder sich dessen Telefonnummer ändert. In diesem Fall muß man die entsprechenden Daten problemlos korrigieren können.

Das Ändern eines Datensatzes, aber auch das Löschen von Datensätzen gehören daher zum Standard jeder Dateiverwaltung. Darüber hinaus erlauben viele Programme auch die Sortierung der einzelnen Datensätze nach einem bestimmten Datenfeld. In unserem Beispiel könnte man die Datei beispielsweise alphabetisch nach den Wohnorten sortieren lassen. Während man mit einer Dateiverwaltung jeweils nur "innerhalb" einer Datei arbeiten kann, ist eine Datenbank, wie Superbase, in der Lage Dateiübergreifend zu arbeiten. Einzelne Datensätze einer Datei, die man zum Beispiel durch bestimmte Suchkriterien "herausgefiltert" hat, lassen sich so leicht in einer anderen oder auch in einer neuen Datei unterbringen.

Dadurch gestaltet sich auch die Umorganisierung bestehender Dateien relativ einfach. Bei einer Dateiverwaltung läßt sich an der Struktur einer Datei in der Regel nichts mehr ändern, ohne daß die bereits eingegebenen Datensätze verloren gehen. Nehmen wir zum Beispiel einmal an, Sie haben eine Adreßdatei mit folgender Eingabemaske definiert:

```
Nachname: <                    >
Vorname:  <                    >

Strasse:  <                    >
Wohnort:  <                    >
```

und vielleicht bereits die ersten 50 Datensätze eingegeben. Und nun stellen Sie fest, daß Sie ja die Telefonnummer vergessen haben! Würde man die Eingabemaske jetzt einfach umändern in:

```
Nachname: <                    >
Vorname:  <                    >

Strasse:  <                    >
Wohnort:  <                    >

Telefon:  <                >
```

dann würden die bereits eingegebenen Datensätze natürlich nicht mehr in dieses Schema passen. Das Gleiche gilt auch für die Länge der einzelnen Datenfelder. Wenn Sie beispielsweise später feststellen, daß Sie das Feld für die Straße zu knapp dimensioniert haben, bleibt Ihnen nur das Abkürzen zu langer Straßennamen oder die Neueingabe der Datensätze. Mit einer Datenbank sind solche Änderungen kein Problem!

Nimmt man sich die Zeit und arbeitet sich in die Programmiersprache einer Datenbank ein (die Programmiersprache von Superbase baut übrigens auf dem BASIC 2.0 des Commodore 64 auf und dürfte daher BASIC-Programmierern kaum Probleme bereiten), so lassen sich sehr komplexe Datenauswertungen realisieren. Nehmen wir wieder ein konkretes Beispiel:

Sie möchten in Ihrem Kleinbetrieb das Rechnungswesen und die Kundenverwaltung vereinfachen. Dazu legen Sie mit Superbase zunächst mehrere Dateien an: eine Adreßdatei mit den Kundenadressen, eine Kontendatei mit sämtlichen Kontenbewegungen sowie eine Lagerdatei, in der die im Lager verfügbaren Artikel gespeichert sind.

Für die einzelnen Verwaltungsvorgänge schreiben Sie sich anschließend individuelle Programme. Ein Programm zum Bearbeiten einer eingehenden Bestellung könnte beispielsweise so aufgebaut sein:

Das Programm erfragt den Namen des Kunden und sieht anschließend in der Adreßdatei nach, ob der Kunde früher schon etwas bestellt hat. Falls nicht, wird seine Adresse in die Adreßdatei aufgenommen und in der Kontodatei ein neues Konto für ihn eingerichtet.

Anschließend erfragt das Programm die bestellten Artikel. Bei jedem Artikel sieht das Programm dann in der Lagerdatei nach, ob der Artikel überhaupt lieferbar ist und gegebenenfalls in welcher Stückzahl. Bei nicht lieferbaren Artikeln gibt es eine entsprechende Meldung aus.

Zum Schluß erstellt das Programm die Rechnung und druckt sie aus. Details wie Mehrwertsteuer, Rabatte und ähnliches, werden dabei automatisch berücksichtigt. Und natürlich vergißt das Programm nicht, den Liefervorgang in der Kontendatei zu vermerken und die Lagerdatei auf den neuesten Stand zu bringen.

Sie sehen, mit einer Datenbank kann man sich ganz individuelle, maßgeschneiderte Problemlösungen erstellen. Gute Programmierkenntnisse sind dazu allerdings schon erforderlich.

2.5 DFÜ

In den Medien war während der letzten Jahre immer wieder die Rede von ihnen: Hackern, die mit Hilfe ihrer Heimcomputer in fremden Datennetzen "herumwildern" und sich unberechtigten

Zugang zu Datenbanken verschaffen. Das Stichwort hierzu heißt Daten-Fernübertragung oder kurz DFÜ. Natürlich möchte ich Sie hier jetzt nicht zu irgendwelchen illegalen Aktivitäten animieren, mit Hilfe der DFÜ lassen sich aber auch auf ganz legale Weise eine Menge interessanter Dinge anfangen.

Das Grundkonzept der Daten-Fernübertragung ist im Grunde genommen sehr einfach: Über das öffentliche Telefonnetz, das ja mittlerweile bis in die entlegensten Winkel der Welt reicht, werden zwei Computer miteinander verbunden, um Daten auszutauschen. Am einen Ende der Leitung befindet sich Ihr Commodore 64, am anderen Ende irgendein anderer Computer. Ob es sich dabei ebenfalls um einen Commodore 64, um einen Personalcomputer oder um einen Großcomputer handelt, ist völlig egal. Spezielle Hardwareeinrichtungen und Programme auf beiden Seiten sorgen dafür, daß es bei der Rechnerkommunikation keine Probleme gibt.

Die Hardware-Einrichtung, die man für den Commodore 64 benötigt, ist entweder ein Akustikkoppler oder ein Modem. Beide Geräte arbeiten die vom Commodore 64 gesendeten Signale derart auf, daß sie über das Telefonnetz geschickt werden können, bzw. bearbeiten die über das Telefon hereinkommenden Signale, damit der 64'er sie verstehen kann.

Ein Akustikkoppler arbeitet, wie der Name schon sagt, auf akustischem Weg. Dazu müssen Sie den Hörer des Telefons auf den Akustikkoppler auflegen. Ein Modem dagegen wird direkt per Kabel an das Telefon angeschlossen, der Umweg über den Telefonhörer entfällt. Ein Modem stellt daher das "professionellere", aber auch teurere der beiden Geräte dar.

Ein Modem garantiert durch die Direktverbindung eine wesentlich bessere Übertragungsqualität. Zwar verfügen Akustikkoppler über spezielle "Muffen", in die der Telefonhörer eingelegt wird. Laute Außengeräusche können diese Muffen aber auch nicht abhalten, so daß es leicht zu Verfälschungen bei der Datenübertragung kommen kann.

Manche Modems sind auch in gewisser Weise "intelligent", d.h. programmierbar. So verfügen viele Geräte über eine automatische Wählfunktion. Eine einmal vom Computer aus eingegebene Telefonnummer wird dann vom Modem so lange (in bestimmten Zeitabständen) angewählt, bis sich der Teilnehmer meldet.

Wie Sie vielleicht wissen ist die Deutsche Bundespost sehr pingelig, was den Anschluß von Fremdgeräten an das öffentliche Telefonnetz betrifft. Wenn Sie sich einen Akustikkoppler oder ein Modem kaufen wollen, sollten Sie deshalb nicht nur auf den Preis achten, sondern auch sichergehen, daß das Gerät über eine sogenannte FTZ-Zulassung verfügt.

Sonst riskieren Sie großen Ärger mit der Post, der sich in hohen Geldstrafen und einer evtl. Beschlagnahmung Ihrer Computeranlage niederschlagen kann. Wenn Sie jetzt vielleicht denken, das Gerät sieht bei mir doch keiner, wie sollte die Post je davon erfahren: Die Post ist in der Lage, Ihren Telefonanschluß vom Fernmeldeamt aus "auszumessen". Dabei kann sie feststellen, ob an Ihrem Telefon irgendwelche Fremdgeräte angeschlossen sind.

Der Anschluß eines Modems oder eines Akustikkopplers an den Commodore 64 bereitet in der Regel kaum Probleme. Es gibt in der Zwischenzeit von verschiedenen Herstellern spezielle Anschlußkabel, mit denen man die Geräte am User-Port des 64'ers anschließen kann.

Mit einem Akustikkoppler oder einem Modem allein können Sie noch nicht viel anfangen. Was Sie noch brauchen, ist die richtige Software. Das beste ist ein sogenanntes Terminal-Programm. Wenn Sie allerdings nur einfache Texte "versenden" wollen, reicht unter Umständen auch ein Textverarbeitungsprogramm. Textomat Plus beispielsweise hat eine spezielle DFÜ-Funktion, mit der man Texte per DFÜ verschicken und empfangen kann.

Mit einem Terminal-Programm läßt sich natürlich wesentlich mehr anfangen. Doch was eigentlich genau?

Da wäre zunächst einmal die Informationsbeschaffung. Es gibt eine ganze Reihe von Informationsdiensten und Datenbanken,

aus denen man per DFÜ Informationen fast jeder Art beziehen kann. Der große Vorteil dabei ist natürlich die Geschwindigkeit, mit der man die Informationen bekommt. Anstatt in eine Bibliothek zu gehen und dort dicke Lexika zu wälzen, was vielleicht einen halben Tag oder mehr beansprucht, hat man die benötigten Informationen aus einer Datenbank meist in Minutenschnelle.

Das hat allerdings auch seinen Preis. Die einzelnen Informationsdienste verlangen zum Teil recht hohe Gebühren. Eine umfangreiche "Recherche" kann da - neben den Telefonkosten - leicht 20 Mark und mehr kosten.

Für den Hobby-DFÜler schon wesentlich interessanter und auch kostengünstiger ist da die Nutzung sogenannter Mailboxen. Bei einer "Mailbox" handelt es sich um eine Art "elektronischem Briefkasten".

Eine Mailbox besteht in der Regel aus einem öffentlichen und einem privaten Teil. Der öffentliche Teil ist für alle Nutzer frei zugänglich. Dort erhält man beispielsweise aktuelle Informationen über wichtige Termine, findet Kauf- und Verkaufsgesuche und zum Teil auch Werbung für verschiedenste Produkte. Das ganze läßt sich noch am ehesten mit einer "Pinwand" oder einem "Schwarzen Brett" vergleichen. Wenn man möchte, kann man dort auch selbst Nachrichten hinterlassen.

Oft möchte man aber irgendeinem Freund eine persönliche Nachricht zukommen lassen, ihm sozusagen einen elektronischen Brief schreiben. Zu diesem Zweck gibt es den privaten Teil der Mailbox. Dort kann man sich einen persönlichen, elektronischen "Briefkasten" einrichten lassen. Dieser Briefkasten ist mit einem Codewort oder einer Codezahl, die nur der betreffende Briefkasteninhaber kennt, geschützt, so daß nicht jeder Mailbox-Anrufer dessen persönliche Briefe lesen kann.

Wenn Sie dem Betreffenden jetzt einen Brief schreiben wollen, senden Sie diesen einfach per DFÜ an die Mailbox, diese legt ihn dann in dessen Briefkasten ab. Sobald Ihr Freund dann selbst

die Mailbox anruft, teilt ihm diese sofort mit, daß ein Brief für ihn eingegangen ist (oder er sieht selbst in seinem Briefkasten nach).

Durch die Arbeit mit einer Mailbox kann man seinen Briefwechsel also wesentlich vereinfachen und beschleunigen, zumal die meisten Mailboxen rund um die Uhr erreichbar sind.

Wem die Arbeit mit Akustikkoppler und Terminalprogramm zu kompliziert ist, der kann sich auf dem Commodore 64 auch eines anderen DFÜ-Mediums bedienen: Bildschirmtext oder kurz Btx. Was Btx ist und was man damit machen kann, dürfte in der Zwischenzeit wohl nahezu jeder wissen. Der Werberummel der Bundespost war und ist schließlich groß genug. Trotzdem werde ich im folgenden kurz darauf eingehen. Die wichtigste Frage ist natürlich zunächst, wie kommt man mit Hilfe des Commodore 64 in den Genuß von Btx?

Ähnlich wie bei der "normalen" DFÜ spielt auch bei Btx das Telefon eine zentrale Rolle. Anstelle eines Akustikkopplers benötigt man einen sogenannten "Btx-Decoder", der von Commodore für etwa 400 Mark angeboten wird. Damit hat man die größte Investition auch schon hinter sich. Der Rest sind Gebühren, zum einen für den Btx-Anschluß durch die Bundespost, zum anderen natürlich für die laufende Nutzung des Btx-Angebotes. An Telefongebühren entstehen grundsätzlich nur Ortstarifgebühren, also DM 0,23 für eine Einheit.

Btx mit dem Commodore 64 hat viele Vorteile. Wenn man Btx mit einem Fernsehgerät einsetzen möchte, entstehen wesentlich höhere Kosten. Allein schon die dafür erforderliche separate Tastatur kostet wesentlich mehr als der ganze Commodore 64, bei dem man die Tastatur ja sozusagen kostenlos mitgeliefert bekommt. Außerdem kann man mit dem 64'er Informationen, die man über Btx bezieht auf Diskette abspeichern und mit einem grafikfähigen Drucker sogar ausdrucken lassen!

Was kann man mit Btx nun konkret anfangen?

Btx wird von einem Computer in Ulm, der sogenannten Leitstelle, zentral gesteuert. Daran angeschlossen sind etwa 50 regionale Vermittlungsstellen, die man in jeder größeren Stadt findet. Die einzelnen Btx-Dienste werden von den verschiedensten, gewerblichen Anbietern bereitgestellt, die dafür auch recht unterschiedliche Gebühren verlangen. Der Kontakt mit Btx läuft immer über eine der regionalen Vermittlungsstellen, die einen gegebenenfalls (etwa bei einer Versandhaus-Bestellung) zu dem jeweiligen Anbieter "durchschaltet".

Das größte Btx-Angebot besteht bei den Informationsdiensten. Die neuesten Aktienkurse lassen sich ebenso abrufen wie aktuelle Fahrplanänderungen der Deutschen Bundesbahn. Das gesamte Angebot ist dabei in einzelne Bildschirmseiten eingeteilt, die direkt über ihre Nummer aufgerufen werden können.

Natürlich gibt es die verschiedensten Hilfen, um ein bestimmtes Angebot zu finden, unter anderem einen Schlagwortkatalog sowie ein Anbieter- und ein Sachverzeichnis. Sucht man beispielsweise ein bestimmtes Warenangebot, etwa Schuhe, so gibt man einfach als Schlagwort "Schuhe" an und erhält dann ein Verzeichnis aller Schuhanbieter, aus dem man auswählen kann.

Damit wären wir auch schon beim zweiten großen Angebotsbereich, dem Bestell-Service. Nicht nur bei den großen Versandhäusern, auch bei zahlreichen kleineren Anbietern kann man über Btx bequem und schnell Waren jeder Art bestellen. Allerdings sinkt erfahrungsgemäß die Hemmschwelle, etwas zu kaufen, auf ein Minimum, je leichter man es dabei hat. Da ist es gut, wenn man sich vorher über seinen aktuellen Kontostand bei der Bank informieren kann. Auch das geht über Btx!

Voraussetzung dafür ist natürlich, daß Ihre Bank einen entsprechenden Service anbietet und Sie mit der Bank eine entsprechende Vereinbarung getroffen haben. Dann können Sie mit Btx sogar Überweisungen von und auf Ihr Konto tätigen.

In diesem Zusammenhang stellt sich natürlich die Frage der Datensicherheit von Btx. Schließlich möchten Sie ja nicht, daß ein Fremder Einblicke in Ihre Finanzen erhält oder gar Geld von Ihrem Konto abbucht. Sehr ärgerlich ist es auch, wenn man Waren geliefert bekommt, die man gar nicht bestellt hat.

Für alle diese Fälle sind in Btx (mehr oder minder) wirksame Schutzmaßnahmen getroffen. Absolute Sicherheit kann es aber nie geben. Wie wirksam der Schutz ist, hängt letztendlich auch von einem selbst ab. Für Banküberweisungen über Btx beispielsweise erhält man von der Bank eine Liste mit mehrstelligen Codenummern, von denen man jede nur einmal benutzen kann. Bei jedem Überweisungsvorgang muß man neben der Bankleitzahl und seiner Kontonummer eine dieser Codenummern zusätzlich angeben. Die Codenummern werden von einem Bankcomputer nach einem speziellen Verfahren berechnet, das nur der Computer selbst kennt.

Ein unüberwindbarer Schutz, könnte man meinen. Doch wenn Sie unvorsichtiger Weise die Liste mit den Codenummern unbeaufsichtigt herumliegen lassen, kann sie leicht in falsche Hände geraten. Wenn der Betreffende dann auch noch Ihre Kontonummer kennt, hat er freien Zugang zu Ihrem Konto. Das ist eben eines der Risiken moderner Kommunikationssysteme, wie Btx, die man gegebenenfalls abwägen muß.

2.6 Lern-Software

Der Bereich "Lern-Software" dürfte wohl mit das schlagkräftigste Argument sein, den Eltern oder Verwandten den Kauf eines Commodore 64 schmackhaft zu machen. Und der Computer als Lernpartner hat ja in der Tat unbestreitbare Vorteile: Er nimmt nichts übel und er ist unendlich geduldig. Gerade, wenn es darum geht, reinen Lernstoff "einzupauken", wie etwa das leidige Vokabeln lernen bei Fremdsprachen, ist der Computer eine interessante Alternative zu herkömmlichen Lernmethoden.

Besonders beim Kauf von Lern-Software sollte man allerdings sehr vorsichtig sein. Der Nutzen eines Lernprogramms steht und

fällt mit den didaktischen Fähigkeiten des Programmierers. Ein Vokabelprogramm, das einen nur auf Fehleingaben aufmerksam macht, aber keine Lösungsvorschläge unterbreitet, ist ebenso nutzlos wie ein Lernprogramm zum Maschinenschreiben, das keine deutschen Umlaute beherrscht.

Die für den Commodore 64 erhältliche Lern-Software hat mittlerweile allerdings ein sehr hohes Niveau erreicht. "Schwarze Schafe" sind nur noch selten anzutreffen. Trotzdem sollte man sich die Programme vor dem Kauf genau anschauen und nach Möglichkeit auch am Rechner vorführen lassen.

Welche Arten von Lern-Software gibt es nun? Alle Lernprogramme lassen sich grob in drei Kategorien einteilen:

Lern-Software für Kinder

Bei dieser Art von Lern-Software steht das spielerische Element im Vordergrund. Einfache Lerninhalte, wie etwa das Bruchrechnen, werden - meist unterstützt durch grafische und musikalische Elemente - spielerisch vermittelt.

Lern-Software für Schüler

Lern-Software für Schüler ist als Ergänzung zum Schulunterricht gedacht. Die Programme orientieren sich daher meist an im Unterricht eingesetzten Lehrbüchern. Im Mittelpunkt stehen hier Programme für Fremdsprachen und Mathematik.

Lern-Software für Erwachsene

Zu dieser Kategorie von Lern-Software zählen zum Beispiel Programme zum Erlernen des Maschinenschreibens oder Programme zum Auffrischen der Allgemeinbildung. Der erste Schritt beim Kauf von Lern-Software sollte es daher sein, festzustellen, für welche Altersgruppe die ins Auge gefaßten Programme geeignet sind. Besonders bei Schul-Software ist auch die angesprochene Zielgruppe (Realschule, Gymnasium, berufsbildende Schule usw...) sehr wichtig.

Und nicht zuletzt sollte man auch das Begleitmaterial einer eingehenden Prüfung unterziehen. Auch ein noch so gut aufgebautes Programm wird mitunter die eine oder andere Frage aufwerfen, die sich dann nur durch ein ausführliches und gut gegliedertes Handbuch klären läßt.

Stellvertretend für die Vielzahl der angebotenen Programme wollen wir uns im folgenden einmal einige Lernprogramme von Heureka Teachware anschauen, die über eine anerkannt gute Qualität verfügen.

Beginnen wir mit einem Lernprogramm für die Grundschule: Der neue Rechenmax. Wie der Name schon vermuten läßt, geht es hier ums Rechnen, genauer gesagt um die vier Grundrechenarten.

Der erste positive Eindruck stellt sich schon beim Laden ein. Das Programm ist durch einen eingebauten Schnel-Lader extrem schnell betriebsbereit. Die Steuerung des Programms erfolgt durchweg über Menüs und ist daher sehr sicher. Positiv fällt auch die mögliche Druckerausgabe auf. Erfolgreich durchgeführte Berechnungen lassen sich so leicht auf Papier "verewigen", indem man einfach eine Hardcopy des Bildschirms macht.

Ein Programm, das nur die vier Grundrechenarten anbietet, wird wohl nichts großes leisten, könnte man meinen. Doch die Autoren des Rechenmax haben sich einiges einfallen lassen.

Zunächst einmal kann man sich vom Rechenmax beliebige Rechenaufgaben stellen lassen, wobei sich der Schwierigkeitsgrad von eins bis sechs individuell einstellen läßt. Sollte man eine Aufgabe einmal nicht selbständig lösen können, so kann man sie sich vom Programm auch vorrechnen lassen. Am Ende der Übungen erhält man dann eine ausführliche Erfolgskontrolle.

Der zweite Teil des Programms orientiert sich am Grundschulunterricht der Klassen eins bis vier. Zu jedem Schuljahr bzw. Halbjahr gibt es passende Übungen.

In der ersten Klasse wird das Zusammenzählen und Zerlegen mit Hilfe von Würfeln und Bällen spielerisch eingeübt. Auch in den Übungen zu den anderen Klassen findet man immer wieder spielerische Aufgaben.

Die Bildschirmdarstellung des Rechenmax kann man als durchweg gelungen bezeichnen, wie überhaupt das ganze Programm. Auf übertrieben aufwendige Grafiken wurde verzichtet. Da alle Abbildungen in sogenannter Blockgrafik erzeugt werden, erfolgt der Aufbau sehr schnell und es entstehen keine unnötigen Wartezeiten.

Das Handbuch zum Programm führt in alle Programmteile in relativ knapper Form ein. Etwas ausführlichere Erläuterungen könnten hier sicher nicht schaden. Da sich das Programm aber sehr anschaulich bedienen läßt, fällt dies nicht weiter ins Gewicht.

Hat man die Grundschule erst einmal hinter sich, so fangen die Probleme mit der Mathematik doch erst so richtig an. Nicht nur die Algebra macht einem zu schaffen, auch die Geometrie hat es in sich.

Zu beiden Bereichen gibt es zwei sehr interessante Programme: GEO-plus und ALI 1001. Wie auch der Rechenmax verfügen beide Programme über einen Floppy-Beschleuniger und eine sehr komfortable Druckerausgabe, was insbesondere bei Geometrieaufgaben natürlich extrem wichtig ist.

Sehen wir uns zunächst einmal ALI 1001 an. ALI ist ein Algebra-Programm, das den Schulunterricht von Klasse 5 bis zur Oberstufe unterstützt.

ALI läßt sich grundsätzlich auf zwei Arten nutzen: als Lernprogramm bzw. Lernspiel und als Werkzeug zum Lösen von Aufgaben. ALIs Fähigkeiten gehen also weit über ein schlichtes Lernprogramm hinaus.

Möchte man mit ALI lernen, so läuft das ganze als Spiel mit oder ohne Punktebewertung ab. Dabei hat man sogar die Mög-

lichkeit, zu zweit zu "spielen"! Nachdem man ein bestimmtes Stoffgebiet ausgewählt und einige weitere Parameter eingestellt hat, kann es losgehen. Da das Programm bei jedem Durchgang mit anderen Zahlen und Formeln arbeitet, läßt sich jeder Programmteil praktisch beliebig oft sinnvoll durcharbeiten. Nur die Lösungswege werden einem mit der Zeit natürlich immer vertrauter. Aber das ist ja auch der Sinn des Ganzen.

Viel interessanter als das Lernen wird den meisten Anwendern die Möglichkeit erscheinen, ALI zum Lösen von Aufgaben einzusetzen. Da das Programm es gestattet, (fast) beliebige Aufgaben einzugeben, kann man ALI also in gewissem Umfang auch zum Lösen der Hausaufgaben einsetzen. Natürlich sollte man die Aufgaben aber zuerst selbst lösen und sich erst anschließend vergewissern, ob man richtig gerechnet hat. Nur so läßt sich ja der gewünschte Lerneffekt erzielen. Und in der Prüfung hat man schließlich auch keinen Commodore 64 zur Verfügung.

Wie sieht das Aufgabenlösen nun konkret aus?

Nehmen wir ein einfaches Beispiel. Sie möchten die folgende Formel berechnen: 648+(731-(69+7*8)/5)/2. ALI geht wie folgt vor:

```
  648+(731-(69+7*8)/5)/2
= 648+(731-(69+56)/5)/2
= 648+(731-125/5)/2
= 648+(731-25)/2
= 648+706/2
= 648+353
= 1001
```

Die Lösung durch ALI erfolgt also schrittweise, so daß das Nachvollziehen keine Probleme bereitet. Auch lineare Gleichungssysteme lassen sich leicht bearbeiten:

```
4(3x+1)-5(x+2) = 3(x-7)-1
  12x-4-5(x+2) = 3x-21-1
   12x-4-5x-10 = 3x-22
        7x-14 = 3x-22
```

```
             4x-14 = -22
                4x = -8
                 x = -2
```

Es macht einfach Spaß, dem Programm bei der Arbeit zuzuschauen! Sogar Bruchterme werden vollständig bearbeitet:

```
3x+7       1   5
-----  =   - - ------
6x+15      2   8x^2-50

N1 : 6x+15   =  3(2x+5)
N2 : 2       =  2
N3 : 8x^2-50 =  2(2x+5)(2x-5)
HN : 6(2x+5)(2x-5)

 D = G \ {-2.5;2.5}

   Multiplikation mit dem Hauptnenner ergibt:

   (3x+7)*2(2x-5) = 1*3(2x+5)(2x-5)-5*3
    2(3x+7)(2x-5) = 3(4x^2-25)-15
     2(6x^2-x-35) = 12x^2-75-15
      12x^2-2x-70 = 12x^2-90
           -2x-70 = -90
              -2x = -20
                x = 10
```

Neben diesen noch relativ einfachen Problembereichen beherrscht ALI auch die Arbeit mit Funktionen aller Art in Vollendung. Dabei wird dann praktisch eine komplette Kurvendiskussion durchgeführt: Die Wertetabelle der Funktion wird erstellt, ihr Schaubild gezeichnet, die Nullstellen bestimmt usw... Das Funktionenspektrum reicht dabei von einfachen Geraden bis hin zu Hyperbeln. Nehmen wir als letztes Beispiel doch einmal eine Nullstellenbestimmung:

```
        x^3-5(3x-4)-10 = -2x(x+1)
         x^3-15x+20-10 = -2x^2-2x
            x^3-15x+10 = -2x^2-2x
       x^3+2x^2-13x+10 = 0

Nullstelle:          x  = 1
                     0
```

```
(x-1)(x^2+3x-10)  = 0
      x^2+3x-10   = 0

                     -3+7
                x  = ---- = 2
                 1     2
D=9+40=49 =>
                     -3-7
                x  = ---- = -5
                 2     2
```

Was bei ALI vor allem überrascht, ist die Flexibilität und Intelligenz mit der das Programm die gestellten Aufgaben löst. Alles in allem ist ALI sicher eine lohnende Anschaffung für jeden Algbra-geschädigten Schüler.

Auch das Programm GEO-plus ist kein Lernprogramm im herkömmlichen Sinne. GEO-plus ist ein Geometrieprogramm, das Schüler der Klassen 7 bis 10 (an Gymnasium und Realschule) beim Konstruieren von Abbildungen und Körpern aller Art unterstützt.

Neben einfachen Dreieckskonstruktionen über geometrische Objekte, wie etwa Geraden und Kreise, lassen sich mit GEO-plus sogar Würfel und Kugeln konstruieren. Besonders interessant ist dabei, daß das Programm zu jeder Konstruktion eine genaue Beschreibung der Vorgehensweise ausdruckt. Anhand dieser Beschreibung kann man die Konstruktion dann leicht selbst auf dem Papier nachvollziehen.

Nehmen wir ein einfaches Beispiel, eine Dreieckskonstruktion, wobei die drei Seitenlängen a,b und c gegeben sind:

```
Beschreibung

Konstruiere ein Dreieck ABC aus :
1. a=5cm, 2. b=6cm und 3. c=7cm

Wähle Punkt A.
Zeichne von A aus die Strecke c.
Bezeichne den Streckenendpunkt mit B.
Zeichne den Kreis k1 um A mit Radius b.
```

```
Zeichne den Kreis k2 um B mit Radius a.
Bezeichne mit C einen Schnittpunkt von k1 und k2.
Zeichne die Strecke a=[B C ].
Zeichne die Strecke b=[A C ].
Dreieck ABC ist das gesuchte Dreieck.
```

Wie ALI, so läßt sich auch GEO-plus hervorragend dazu "mißbrauchen", die Hausaufgaben zu lösen, was man natürlich nur im Notfall tun sollte! Denn schließlich: Übung macht den Meister. Und die kann einem kein auch noch so gutes Lernprogramm abnehmen.

Im Gegensatz zu den bisher vorgestellten Programmen geht es bei der Reihe "Learning English" ums reine Lernen, genauer gesagt um das Vokabel-Lernen. "Learning English" ist an die gleichnamige Buchreihe aus dem Klett-Verlag angelehnt und besteht aus insgesamt sechs Teilen, die separat erhältlich sind. Jeder Teil umfaßt etwa 1000 Vokabeln, die auf vielfältige Art und Weise eingeübt werden können.

Die Handbücher der einzelnen Teile sind sehr knapp dimensioniert und beschränken sich auf eine Beschreibung der Bedienung der Programme. Zwar kann man mit den Programmen auch so sehr gut arbeiten, doch empfiehlt es sich trotzdem die entsprechenden Klett-Bücher zu kaufen. Mit beiden zusammen läßt sich ideal arbeiten.

Alle Programme dieser Reihe sind analog aufgebaut. Zunächst wählt man die Vokabeln aus, mit denen man arbeiten möchte. Das können die Vokabeln einer kompletten Unit sein oder auch nur Teile daraus.

Anschließend kann man festlegen, ob die Vokabeln isoliert oder im Kontext, d.h. eingegliedert in Übungssätze, eingeübt werden sollen. Hat man sich für die isolierte Abfrage entschieden, so gibt es auch hier noch mehrere Möglichkeiten:

Vorgabe: deutsche Vokabel - Gesucht: englische Vokabel
Vorgabe: englische Vokabel - Gesucht: deutsche Vokabel
Vorgabe: englische Definition - Gesucht: englische Vokabel
Vorgabe: Zufallsprinzip - Gesucht: engl./deutsche Vokabeln

Schließlich läßt sich auch noch einstellen, in welcher Reihenfolge die Vokabeln abgefragt werden sollen.

Diese vielfältigen Einstellmöglichkeiten zeugen schon von der Qualität der Programme. Den größten Nutzen bzw. Lerneffekt bringen die Programme aber durch ihre ausgeklügelte Abfragetechnik.

Eine Vokabel, die falsch eingegeben wurde, wird intern im Programm markiert und später (während der Übung) zweimal erneut abgefragt. Erst wenn man sie zweimal richtig eingegeben hat, wird sie als "gekonnt" gespeichert. Vokabeln, die man schon beim ersten Mal vollständig korrekt eingegeben hat, werden später nicht noch einmal abgefragt. Dadurch ist sichergestellt, daß man ohnhin schon bekanntes nicht ständig unnötig wiederholt.

Wer eine zusätzliche Hilfe benötigt, kann sich auch den ersten und letzten Buchstaben einer gesuchten Vokabel vorgeben lassen. Daß man am Ende der Übung eine ausführliche Lernstatistik erhält, versteht sich schon fast von selbst.

Alles in allem ist "Learning English" also sehr empfehlenswert. Der einzige Kritikpunkt ist die fehlende Möglichkeit, den Vokabelschatz zu erweitern. Dieser Mangel hängt aber wohl mit dem konzeptionellen Aufbau der Reihe zusammen. Jedes Programm ist ja als Ergänzung zu dem zugehörigen Klett-Buch gedacht.

Wem das reine Vokabelnlernen zu stupide ist, der sollte sich einmal die englischen Sprachübungen 4-6 von Heureka Teachware anschauen. Diese Übungen bestehen aus drei recht originellen Lernspielen. Im ersten Spiel "Caught in the Castle" beispielsweise geht es darum, einen entführten Geheimagenten aus einem alten Schloß zu befreien.

In allen Lernspielen muß man einen Lückentext ergänzen. Bei "Caught in the Castle" sieht das zum Beispiel so aus: Man befindet sich in einer bestimmten Situation, etwa in einem Raum mit einem Schrank, und hat nun mehrere Möglichkeiten:

```
Options:
     (a) look into the wardrobe
     (b) open the window
     (c) turn the light on

Possible consequences:
     (1) a helicopter comes along
     (2) the butler brings a drink
     (3) an alarm bell rings outside
```

Gibt man nun beispielsweise <a> und <1> ein, erscheint folgender Lückentext,

```
if he .....
into the wardrobe,
a helicopter .....
```

den man durch "looked" und "would come" ergänzen muß.

Hier geht es also weniger um das Lernen neuer Vokabeln, sondern mehr um das Einüben der englischen Grammatik. Durch die Vielzahl der Kombinationsmöglichkeiten (im obigen Beispiel könnte man ja auch <b> und <3> nehmen) bleibt der Spielwert und damit auch der Lernwert der Programme lange erhalten.

2.7 Spiele

Wohl für keinen anderen Computer gibt es ein derart umfangreiches Angebot an Spielen wie für den Commodore 64. Und manch einer wird sich den Commodore 64 wohl vor allem gekauft haben, um mit ihm zu spielen. In diesem Abschnitt möchte ich Ihnen deshalb einmal anhand bewährter Spiele einen Überblick geben, welche Spielgattungen auf dem 64'er vertreten sind. Das Angebot beschränkt sich nämlich bei weitem nicht nur auf sogenannte "Ballerspiele". Neben zum Teil recht aufwendigen Simulationen findet man auch immer mehr Gesellschaftsspiele, mit denen man zu zweit oder mit mehreren Personen spielen kann. Ebenfalls sehr beliebt sind sogenannte Adventures und Rollenspiele.

Kein Markt ist so schnellebig wie der Spielemarkt. Die meisten Spiele sind nur wenige Wochen im offiziellen Verkauf, dann kommen schon die nächsten. Da fällt es natürlich umso schwerer, zu entscheiden, ob ein neues Spiel das hält, was die Werbung verspricht. Diesen Umstand haben sich einige Zeitschriftenverlage zunutze gemacht. Es gibt eine ganze Reihe von Zeitschriften, die sich fast ausschließlich mit dem Testen der neuesten Spiele befassen. Wer sich öfters neue Spiele kaufen möchte, sollte solche Spielezeitschriften, die man an jedem größeren Kiosk findet, ruhig regelmäßig lesen; manch teurer Mißgriff läßt sich dadurch vermeiden.

Bei den folgenden Spielebeschreibungen habe ich zum Großteil auf bewährte "Klassiker" zurückgegriffen, die trotz ihres zum Teil schon erheblichen Alters nichts von ihrem Spielwert eingebüßt haben.

Da die Preise von Händler zu Händler mitunter sehr verschieden sind, habe ich auf Preisangaben bewußt verzichtet. Ganz allgemein liegen die Preise etwa zwischen 10 und 80 Mark.

Viele ältere Spiele gibt es in der Zwischenzeit auch im Sammelpaket, d.h. mehrere Spiele (manchmal bis zu zehn) zum Preis von einem. Da lohnt sich das Nachfragen beim Händler.

Chuck Yeager's AFT

Chuck Yeager's Advanced Flight Trainer ist eine sehr abwechslungsreiche Flugsimulation. Nicht weniger als 14 verschiedene Flugzeuge stehen einem zur Verfügung, von einer kleinen Cessna bis hin zu einem F-16-Jet ist alles vorhanden. Obwohl auch Militärmaschinen dabei sind, ist das Spiel durchweg friedlich. Auf das "Durchexerzieren" irgendwelcher Kampfszenen wurde lobenswerterweise gänzlich verzichtet. Um was es geht, sind Wett- und Formationsflüge. Wer will, kann zunächst auch in eine "Flugschule" gehen, um dort die Grundtechniken des Fliegens zu lernen. Auf dem Programm stehen Starts und Landungen, Geradeausflug, Kurven und sogar Loopings. Dabei macht es natürlich einen Riesenunterschied, ob Sie mit einer Cessna oder einer F-16 einen Looping drehen.

Bei einem Wettflug geht es darum, gegen einen anderen vom Computer simulierten Piloten als erster eine Slalomstrecke zu durchfliegen. Sechs unterschiedliche Strecken stehen dazu zur Auswahl.

Beim Formationsfliegen müssen Sie versuchen, möglichst genau hinter einem anderen Flugzeug her zu fliegen. Für Abweichungen von der Flugbahn gibt es Abzüge. Mit Hilfe des integrierten Flugrekorders lassen sich auch eigene Flüge aufnehmen, so daß man anschließend sozusagen sich selbst hinterherfliegen kann. Selbstkreierte Formationen kann man so sehr leicht einüben und später dann seinen Freunden "vorfliegen".

Deactivators

Deactivators ist eine Mischung aus Denkspiel und Actionspiel, wobei beide Spielgattungen gleichberechtigt nebeneinader stehen.

In insgesamt fünf Gebäuden haben Terroristen einige Bomben gelegt, die in kurzer Zeit explodieren können. Ihre Aufgabe ist es nun, diese Bomben mit Hilfe von Spezialrobotern aus den Gebäuden zu schaffen.

Das schwierigste an der ganzen Aufgabe ist es nun nicht, die Bomben zu finden, sondern die Roboter überhaupt heil durch die Gebäude zu bringen. Es gibt nämlich nicht nur allerlei Wächter, die den Robotern nach dem "Leben" trachten, auch die Schwerkraftverhältmnisse in den einzelnen Räumen haben ihre Tücken. In manchen Räumen befindet sich "unten" an einer der Seiten oder gar an der Decke. Auch sind nicht alle Räume durch eine Tür verbunden.

Da das ganze gegen die Zeit läuft (eine der Bomben könnte ja jeden Augenblick explodieren), ist für ausreichenden Nervenkitzel gesorgt.

Elite

Das Spiel Elite kann man als Klassiker schlechthin bezeichnen. Elite ist ein kombiniertes Strategie-Action-Spiel, das im Welt-

raum spielt. Als intergalaktischer Händler müssen Sie nicht nur kaufmännisches Geschick beweisen, sondern sich auch gegen allerlei Gefahren, beispielsweise Raumpiraten, erwehren.

Elite ist recht leicht zu erlernen, dafür aber ungeheuer komplex. Selbst gute Spieler brauchen meist Monate, um "Elite-Kämpfer" zu werden.

Obwohl bei Elite auch die Action nicht zu kurz kommt, steht das strategische Element im Mittelpunkt. Es gilt vor allem, sein Handelsgeschick unter Beweis zu stellen. Zu Beginn starten Sie mit Ihrem Raumschiff, ausgerüstet mit einem "Startkapital" von 100 Credits in eine von acht Galaxien. Das höchste Ziel ist es, Elite-Kämpfer zu werden.

Elite beeindruckt am Bildschirm durch eine sehr schnelle 3D-Vektorgrafik. Bei der Vektorgrafik werden nur die Umrisse eines Objektes gezeichnet. Ein Würfel beispielsweise wird nur durch seine Kanten dargestellt. Die Flächen dazwischen sind "durchsichtig".

Durch seinen hohen Spielwert und die Faszination, die es ausübt, ist Elite mit Sicherheit eine lohnende Investition.

Gauntlet

Bei Gauntlet handelt es sich um die Umsetzung eines sehr erfolgreichen Spielautomaten. Gauntlet ist ein Rollenspiel mit ausgeprägten Action-Elementen.

Bei Gauntlet streift man durch die Gewölbe einer unterirdischen Welt und ist - was sonst - auf der Suche nach allerlei Schätzen.

Das besondere dabei: Zwei Spieler arbeiten im Team zusammen. Obwohl an sich jeder Spieler darauf aus ist, möglichst viele Punkte zu machen, kommt man in vielen Situationen nur durch "Teamwork" weiter. Beeindruckend sind nicht zuletzt auch die zahlreichen animierten Gegner (zum Teil Dutzende), die sich gleichzeitig auf dem Bildschirm befinden.

Ghosts'n Goblins

Ghosts'n Goblins ist eine sehr gelungene Mischung aus Adventure und Action-Spiel. Die Handlung spielt im Mittelalter. Einem tapferen Ritter ist die liebe Braut abhanden gekommen; sie wurde von einem bösen Dämon entführt. Was folgt, ist klar: Der Ritter versucht, das Mädchen zu befreien. Daß der Dämon damit nicht einverstanden ist, dürfte ebenso klar sein. Der Ritter muß sich nun in den verschiedensten Szenarien gegen allerlei Monster, Geister, Dämonen und fleischfressende Pflanzen erwehren. Dazu stehen ihm mehrere Waffen zur Verfügung.

Das gesamte Spiel besteht aus insgesamt vier Levels. Jeder Level ist in acht Stufen (Wald, Geisterstadt, Eispalast usw...) unterteilt. Um von einer Stufe in die nächste zu kommen, muß man zunächst einen kleinen Dämon beseitigen. Die Tür zum nächsten Level bewacht ein großer Dämon, für den schon etwas mehr Kampfkraft erforderlich ist. Hat man sich dann endlich durch alle Level durchgekämpft, bleibt nur noch ein letzter Gegner, ein chinesischer Drache, der die Braut bewacht.

Hier noch ein Tip:

Laden Sie das Programm ganz normal. Wenn Sie das Titelbild sehen, drücken Sie den RESET-Taster (der allerdings schon eingebaut sein muß) und geben einen der beiden POKE ein.

```
POKE 4170,10   REM Sprite-Kollision ausschalten
POKE 2756,255   REM 175 Leben
```

Anschließend startet man das Spiel wieder mit SYS 2128.

Gunship

Bei Gunship handelt es sich um eine sehr aufwendige Flugsimulation. Simuliert wird der Hubschrauber AH-64 A, ein Kampfhubschrauber. Daher geht es im gesamten Programm recht kriegerisch zu. Die Hauptaufgabe ist aber weniger, irgendwelche Feinde zu vernichten, sondern vielmehr, den Hubschrauber zu beherrschen.

Dazu werden von Gunship zahlreiche Funktionen eines echten Hubschraubers simuliert. Mit Hilfe von Joystick und Tastatur gilt es, den Hubschrauber auf Kurs zu halten. Außerdem muß man die verschiedenen Waffen- und Abwehrsysteme korrekt bedienen, um sich Gegner vom Hals zu halten.

Blickt man auf den Bildschirm, so fühlt man sich in ein Hubschrauber-Cockpit versetzt. Neben den diversen Instrumentenanzeigen erhält man einen Blick auf die unter einem liegende Landschaft.

Um möglichst realitätsnah zu sein, erlaubt Gunship das Fliegen von Missionen in "echten" Krisengebieten. Die Szenarien reichen dabei von Einsätzen in der USA bis zu Kämpfen im Mittleren Osten. Sicher nicht jedermanns Sache. Es werden allerdings keine konkreten Vorfälle nachgespielt. Die einzelnen Szenarien unterscheiden sich - neben der Landschaft, über die man fliegt - nur in der Stärke und Bewaffnung des jeweiligen Gegners.

Impossible Mission

Impossible Mission kann man als Mischung aus Strategie-, Action- und Geschicklichkeitsspiel bezeichnen. Neben einem spannenden und abwechslungsreichen Spielprinzip verfügt Impossible Mission über eine hervorragend animierte Grafik, die fast an einen Zeichentrickfilm erinnert, und einer recht gut gemachten Sprachausgabe. Die Handlung ist eher banal (als Mitglied einer Antiterrorgruppe müssen Sie einen potentiellen Weltbeherrscher namens Elvin ausschalten), der Weg zu diesem Ziel dafür aber umso interessanter.

Es gilt, einen Weg durch die Räume und Tunnel des unterirdischen Hauptquartiers von Elvin zu finden, um bis zu ihm vorzudringen. Dabei hat man natürlich mit allerlei Gefahren zu kämpfen. Außerdem muß man sich die einzelnen Buchstaben eines Paßwortes zusammensuchen, die auf insgesamt 36 Puzzleteilen vermerkt sind, die in den verschiedenen Räumen wild verstreut liegen. Dieses Paßwort benötigen Sie später für den Kontrollraum von Elvin.

Das Haupthindernis sind die Roboterwachen, die man auf verschiedene Art und Weise überlisten kann. Eine Möglichkeit ist das Aufspüren spezieller Hilfspaßworte, die in einigen Räumen zu finden sind. Man kann sie sich aber auch durch einen Beweis seiner musikalischen Fähigkeiten verschaffen.

In dem Labyrinth gibt es nämlich zwei Räume, in denen sich eine Computerkonsole und eine Anzeigetafel befinden. Wenn Sie vor der Konsole stehen und den Joystick nach vorne bewegen, hören Sie drei Töne, zu denen jeweils ein Feld der Anzeigetafel aufleuchtet. Danach erscheint eine Hand, mit der Sie die Felder anfahren können. Es geht nun darum, die Töne vom tiefsten bis zum höchsten nachzuspielen. Dazu müssen Sie die stilisierte Hand mit dem Joystick auf das betreffende Feld positionieren und anschließend den Feuerknopf drücken. Falls Sie die richtige Reihenfolge eingehalten haben, erhalten Sie ein Hilfspaßwort und das ganze beginnt von vorne, diesmal aber mit einem Ton mehr.

Koronis Rift

Bei Koronis Rift handelt es sich um ein Strategie-Action-Spiel. Im Gegensatz zu Elite steht aber eindeutig das Action-Element im Vordergrund.

Als galaktischer Glücksritter sind Sie auf der Suche nach den Überresten vergangener Kulturen. Ziel ist es, neue technische Geräte zu finden und diese an Wissenschaftler zu verkaufen.

Bei Ihrer Suche stoßen Sie früher oder später auf einen Planeten namens Koronis Rift. Koronis Rift wurde vor vielen Jahren als Testgelände für neue Waffensysteme benutzt. Nun geht es darum, dem Planeten diese technologischen Geheimnisse zu entreißen. Doch dazu müssen zunächst einmal die Abwehrsysteme von Koronis Rift überwunden werden.

Dabei kommt dann auch das strategische Geschick nicht zu kurz. Auf dem Planeten gibt es nämlich über 150 "Module", mit denen man die Schlagkraft seines Raumschiffs verbessern kann. Es

lassen sich aber immer nur maximal sechs Module gleichzeitig einsetzen. Die richtige Auswahl der Module trägt daher entscheidend zum Kampferfolg bei.

Nemesis

Nemesis ist ein Action-Spiel im klassischen Sinn. Hier geht es wirklich nur ums "drauflosballern". Es gilt, den Planeten Nemesis von den bösen Bakteroiden zu befreien. Dazu gibt es insgesamt acht Level, wobei man es in jedem Level mit einer anderen Art von Gegner zu tun hat.

Einmal greifen einen bis zu acht Raumschiffe gleichzeitig an, ein anderes Mal muß man sich gar gegen Skelette erwehren. Hat man einen Gegner vernichtet, so erscheint ein gelber "Energiepunkt", den man aufsammeln sollte. Hat man genügend Punkte beisammen, dann erhält man eine Extrawaffe, mit der sich den Gegnern noch besser zu Leibe rücken läßt.

Nach dem letzten Level bekommt man es schließlich noch mit dem "Ober-Bakteroiden" zu tun. Sobald auch dieser überwunden ist, geht das ganze wieder von vorne los.

Wer ein reines Schießspiel sucht, ist mit Nemesis sicher bestens bedient.

Ein kleiner Tip: In Nemesis kommt man leicht in den sogenannten Trainermodus, erhält also "unendliches" Leben. Dazu drücken Sie einfach, während Sie sich im freien Weltraum befinden, die Tasten <J>, <K> und <L> gleichzeitig. Wem das noch nicht genügt, sollte einmal folgende Zeile ausprobieren (nachdem er das Spiel durch einen RESET unterbrochen hat):

```
POKE 5975,234:POKE 5976,234:POKE 5977,234:SYS 5779
```

Paradroid

Paradroid ist eine Kombination aus Action- und Strategiespiel, wobei die Action allerdings leichten Vorrang hat. In einer Flotte von acht Raumschiffen haben die Roboter begonnen, verrückt

zu spielen. Um die Raumschiffe zu retten, bleibt nur die Möglichkeit, einen Spezialroboter an Bord der Schiffe zu bringen. Dieser Spezialroboter mit der Nummer 001 wird von der Erde aus bzw. von Ihnen als Spieler ferngesteuert.

Um die verrückt gewordenen Roboter zu bekämpfen, verfügt 001 natürlich über die obligatorische Laserkanone, aber er hat auch noch eine besondere Fähigkeit: Er ist in der Lage, sich mit einem speziellen Interface an die verschiedenen Roboter anzuschließen und diese dann zu beeinflußen.

Der Anschluß ist allerdings nicht ganz einfach. Er läuft jeweils - jetzt kommt das strategische Element ins Spiel - über ein kleines Logikspiel, das man lösen muß. Außerdem gibt es insgesamt 30 verschiedene Robotertypen, die alle über unterschiedliche Eigenschaften verfügen. Da muß man sich schon sehr genau überlegen, an welchen Roboter man sich "dranhängt", um zum Ziel zu kommen.

Parallax

Parallax ist ein kombiniertes Action- und Adventure-Spiel, wobei die Action-Elemente eindeutig im Vordergrund stehen. Die Handlung ist recht simpel gestrickt. Es geht wie in den meisten Action-Spielen gegen böse Außerirdische, die die Erde erobern wollen.

Alles beginnt damit, daß die Außerirdischen fünf Astronauten auf ihr Raumschiff entführen und dort in fünf getrennten Sektoren gefangenhalten. Sie als Spieler sind einer dieser fünf Astronauten und sollen nun versuchen, die vier anderen zu befreien und dann das Raumschiff zu zerstören. Doch das ist leichter gesagt als getan.

Zunächst gilt es, sich einen speziellen Öffnungs-Code zu besorgen, um das Sicherungssystem des Raumschiffs zu überwinden. Auch im weiteren Verlauf des Spiels sind immer wieder Codes zu ermitteln, um voranzukommen. Parallax geht also um einiges über ein reines "Ballerspiel" hinaus.

Vorausgesetzt man hat einen RESET-Schalter für seinen 64'er, so kann man sich das Spielen mit Parallax etwas leichter machen. Starten Sie dazu das Programm, und führen Sie dann einen RESET herbei. Anschließend geben Sie folgende Zeile ein und drücken die <Return>-Taste:

```
POKE 5796,96:SYS 319
```

Spindizzy

Spindizzy ist ein Geschicklichkeitsspiel. In einer anderen Dimension wurde ein rätselhaftes Gebilde entdeckt, das nun von Ihnen "kartographiert" werden soll. Zur Unterstützung erhalten Sie ein kreiselförmiges Fahrzeug namens Gerald.

Doch Gerald braucht Energie. Um diese bezahlen zu können, gilt es, möglichst schnell neue Teilabschnitte des Gebildes zu entdecken, denn nur dafür gibt es Geld, für Gerald neue Energie und damit für Sie erweiterte Spielzeit. Das gesamte Gebilde besteht aus rund 400 Teilabschnitten, wobei jeder Teilabschnitt den ganzen Bildschirm beansprucht. Da ist für langen Spielspaß gesorgt.

Neben Treppen, Brücken, Seen, Eisflächen und ähnlichem mehr gilt es vor allem, ein System von Schaltern und Liften zu überwinden. Bevor Sie einen Lift benutzen können, müssen Sie diesen zuerst mit einem Schalter aktivieren. Diese Schalter sind aber oft durch Hindernisse blockiert, die sich nur mit anderen Schaltern beseitigen lassen. Es dürfen aber immer nur zwei Schalter gleichzeitig eingeschaltet sein. Hier ist logisches und kombinatorisches Geschick gefragt.

Ein kleiner Tip am Rande: Um für Ihre Erkundungen unbegrenzte Zeit zu erhalten, geben Sie direkt nach dem Laden des Spiels einfach "PAT" ein.

The Bard's Tale

The Bard's Tale ist ein Adventure in der Form eines Rollenspiels. Eine Stadt wurde von einem bösen Zauberer verflucht.

Sechs Abenteurer wollen versuchen, die Stadt von dem Zauberer zu befreien. Ihre Aufgabe ist es nun, diese sechs Abenteurer zu steuern.

Dazu müßen Sie zu Beginn für jeden von ihnen zunächst seine Eigenschaften und Fähigkeiten festlegen, etwa, ob er magische Kräfte haben oder doch lieber mit einem Schwert kämpfen soll. Die Auswahlmöglichkeiten dabei sind natürlich begrenzt. Die richtige Kombination der einzelnen Fähigkeiten ist daher entscheidend für den späteren Erfolg.

Anschließend kann der Kampf gegen den Zauberer aufgenommen werden. Ziel ist es, bis zum Zauberer vorzudringen und diesen dann auszuschalten. Dazwischen stehen insgesamt sechszehn zum Teil sehr große Labyrinthe und zahlreiche Schlösser, Verliese, Abwasserkanäle und ähnliches. Natürlich sind auch zahlreiche Rätsel zu lösen, und die Abenteurer müssen gegen die Gehilfen des Zauberers kämpfen.

Das ganze Spiel wird übrigens über die Tastatur gesteuert, der Joystick ist nicht gefragt. Das Spiel gibt Ihnen dazu jeweils einen aktuellen "Situationsbericht", anschließend müssen Sie irgendwie reagieren, etwa einen Zauberspruch sprechen oder an einen der sechs Abenteurer ein Kommando geben.

The Great Giana Sisters

The Great Giana Sisters ist ein Geschicklichkeitsspiel, genauer gesagt ein "Hüpfspiel" der ganz besonderen Art. Auf insgesamt 32 verschiedenen Levels gilt es, der kleinen Giana zu helfen, den Riesendiamanten zu finden. Daß das kein leichtes Unterfangen ist, versteht sich natürlich von selbst.

Zu Beginn ihrer Suche hat Giana vier Leben. Durch das Anspringen von mit einem Stern markierten Felsbrocken kann sie Spezialfähigkeiten erwerben, etwa die Fähigkeit, Steine zu zerschlagen oder bestimmte Kreaturen in einen kurzen Schlaf zu versetzen. Mit Hilfe dieser Fähigkeiten läßt sich manches Hindernis aus dem Weg räumen bzw. umgehen.

In den meisten Fällen verbirgt sich hinter den Felsbrocken aber ein Diamant, der ihre Widerstandskraft erhöht. Hat sie 100 Diamanten eingesammelt, erhält sie sogar ein neues Leben. Verliert sie ein Leben, so gehen damit automatisch auch alle ihre Spezialfähigkeiten verloren. Wem das Spiel zu schwer ist, der sollte einmal - nach einem RESET - die folgende Zeile eingeben (<Return>-Taste drücken nicht vergessen!):

```
POKE 2446,255:POKE 6697,255:SYS 2098
```

The Guild of Thieves

The Guild of Thieves, zu deutsch "Die Gilde der Diebe", ist ein sogenanntes Grafik-Adventure, ein Abenteuerspiel also. Die insgesamt 30 Grafiken des Spiels, die über eine sehr gute Qualität verfügen, dienen mehr der Verzierung, man könnte das Spiel auch ohne sie spielen.

Das eigentliche Spiel läuft rein über Textausgaben und Texteingaben. Da das Spiel aus England kommt, sind leider gute Englischkenntnisse erforderlich. Doch, um was geht es eigentlich? In einem fernen Land namens Kerovnia haben sich einige Einbrecher und Räuber in einer Gilde zusammengeschlossen, in die Sie als Spieler aufgenommen werden wollen. Dazu müssen Sie eine Aufnahmeprüfung bestehen.

Es gilt, aus einem alten Schloß verschiedene Schätze zu entwenden. Doch dazu muß man sie natürlich erst einmal finden. Und auch dann ist oft noch manches Problem zu lösen, bevor man den Schatz tatsächlich mitnehmen kann.

The Pawn

The Pawn zählt wie "The Guild of Thieves" zur Kategorie der Grafik-Adventures. Faszinierende Grafiken ergänzen eine gut durchdachte und sehr komplexe Handlung. Leider erfordert The Pawn gute Englischkenntnisse und natürlich, wie bei jedem Adventure, jede Menge Geduld und Ausdauer.

Bestandteil des Programms ist ein knapp 50-seitiger Roman, der zum Lösen des Spiels unbedingt erforderlich ist. Der Roman enthält am Ende auch zusätzliche Hilfen zu den einzelnen Rätseln, allerdings in codierter Form. Man muß die Lösungstips erst mit Hilfe des Spiels decodieren. Zu Beginn des Spiels verschlägt es einen in das Land Kerovnia, in dem es nicht nur politisch drunter und drüber geht. Auch allerlei Zauberer treiben ihr Unwesen. Das Ziel ist es, wieder aus Kerovnia herauszukommmen. Doch, wie das zu bewerkstelligen ist, erfährt man erst so nach und nach im Laufe des Spiels.

Neben der exzellenten Grafik beeindruckt vor allem der schnelle Parser, der auch komplizierte Sätze versteht. Sie wissen nicht, was ein Parser ist? Ein Parser dient bei allen Adventures dazu, den vom Spieler eingegebenen Text zu analysieren. Je nach der Qualität des Parsers verfügt dieser über einen mehr oder minder großen Wortschatz und kann auch kompliziertere Sätze verstehen. Mit dem Parser von The Pawn kann man sich schon fast in Umgangssprache (wenn auch in englischer) "unterhalten".

The Sentinel

The Sentinel ist ein Action-Strategie-Spiel der ganz besonderen Art. In einer fremden Dimension, die aus insgesamt 10.000 Welten besteht, herrscht der Sentinel. Sie sollen diese Welten nun in der Rolle eines Roboters vom Sentinel befreien.

Das Spielprinzip ist an sich recht einfach: Jede Welt ist in einzelne Quadrate eingeteilt. Jedes Quadrat hat eine bestimmte Höhe. Wenn man sich auf einem bestimmten Quadrat befindet, kann man immer nur Quadrate beeinflussen, die sich "unter" einem befinden.

Zu Beginn des Spiels befindet sich der Sentinel natürlich ganz oben und Sie ganz unten am niedrigsten Punkt. Nun gilt es, sich bis auf die Höhe des Sentinel hinaufzuarbeiten und diesen dann zu vernichten. Der Sentinel ist damit natürlich überhaupt nicht einverstanden und versucht, Sie nach Kräften daran zu hindern.

The Sentinel ist ein Spiel, mit dem man dank der vielen verschiedenen Welten monatelang spielen kann. Man kann sich das ganze aber auch etwas vereinfachen. Probieren Sie doch einmal die folgenden POKEs. Dazu starten Sie das Spiel und führen dann einen RESET herbei. Anschließend geben Sie die folgende Zeile ein und drücken die <Return>-Taste:

```
POKE 6679,173:POKE 8512,10:SYS 16128
```

Trailblazer

Trailblazer ist ein Geschicklichkeitsspiel mit recht ungewöhnlicher Handlung und einem interessanten Spielprinzip. Trailblazer arbeitet mit einem "gesplitteten" Bildschirm, d.h., der Bildschirm ist horizontal in zwei Hälften geteilt. In beiden Hälften kann jeweils ein Spieler völlig unabhängig vom anderen spielen.

Die Aufgabe bei Trailblazer besteht darin, einen kleinen Ball möglichst schnell über eine Art "Fließband" zu bewegen. Dieses Fließband ist in verschiedenfarbige Karomuster unterteilt und hat auch einige Löcher, durch die der Ball hindurchfallen kann. Manche der Karofelder (erkenntlich an ihrer Farbe) bremsen den Ball ab, manche beschleunigen ihn. Darüber hinaus sind noch einige andere "Hindernisse" eingebaut, die es zu überwinden gilt. Wie alle Wettrennen macht Trailblazer am meisten Spaß, wenn man es zu zweit spielt. Man kann aber auch allein gegen den Computer antreten.

Trivial Pursuit

Trivial Pursuit ist ein Spiel, das ausnahmsweise einmal für die ganze Familie geeignet ist. Viele werden das gleichnamige Brettspiel kennen, das für dieses Spiel als Vorlage diente. Trivial Pursuit orientiert sich recht genau an den Regeln seiner Vorlage. Wem also Brettspiele zu altmodisch sind, der hat mit Trivial Pursuit als Computerspiel eine interessante Alternative. Für all diejenigen, die Trivial Pursuit noch nicht kennen, ein paar Worte zum Spielprinzip:

Trivial Pursuit ist ein Quizspiel mit zum Teil sehr ungewöhnlichen, aber in der Regel einfach zu beantwortenden Fragen. Der Spielplan besteht aus etwa 100 Spielfeldern, die sechs verschiedene Farben haben. Jede Farbe steht für ein bestimmtes Wissensgebiet, etwa Wissenschaft oder Unterhaltung.

Bei jedem Spielzug muß man eine Frage aus dem zu der betreffenden Farbe des Feldes gehörenden Wissensgebiet beantworten. Nur wenn man die Frage richtig beantwortet, darf man weiterspielen. Ansonsten ist der nächste Spieler dran. Insgesamt 3000 Fragen stehen zur Verfügung, wobei der Computer die gestellten Fragen nach dem Zufallsprinzip auswählt. Viele Fragen werden nicht nur als reine Textfragen gestellt, sondern haben eine Grafik- oder Soundkomponente. Dadurch wird das ganze im Vergleich zum Brettspiel um einiges abwechslungsreicher.

World Games

World Games zählt zur Kategorie der Sportspiele. Entweder allein oder mit bis zu sieben Mitspielern kann man an insgesamt acht, zum Teil recht exotischen Wettkämpfen teilnehmen und sein sportliches Talent unter Beweis stellen. Den meisten Spaß bringt das Spiel natürlich, wenn man es mit möglichst vielen "Athleten" zusammen spielt.

Die Sportarten sind wie gesagt recht exotisch. Da geht es beispielsweise zum Sumo-Ringkampf nach Japan, zum Gewichtheben nach Rußland oder zum Bierfaß-Springen nach Deutschland.

Die Steuerung erfolgt bei allen Wettkämpfen über den Joystick. Mal muß man den Joystick sekundengenau in eine bestimmte Richtung bewegen, mal genau im richtigen Augenblick den Feuerknopf betätigen. Das ist zwar auf Dauer nicht übermäßig originell, aber es macht einfach Spaß, die Reaktionen der Spielfiguren, die sehr detailreich dargestellt und animiert sind, am Bildschirm zu beobachten.

Nach jedem Wettkampf werden die drei besten Ergebnisse mit einer Medaille belohnt. Am Ende wird daraus ein Gesamtsieger ermittelt.

2.7.1 Spielehilfen

Dieser Abschnitt gibt Ihnen kurze Hilfen, wie Sie ein Spiel besser bewältigen können. Los geht's!

Arkanoid

Wußten Sie schon, daß es in Arkanoid einen Trainermodus gibt? Stellen Sie am Anfang einmal folgende Parameter ein:

```
2 Players / 1 Joystick / 1 Device.
```

Der erste Spieler hat nun ein ganz normales Spiel. Der zweite Spieler befindet sich im Trainermodus, sobald er 20000 Punkte erreicht hat. Dann erhöht sich nämlich sein Lebenskonto bei jeder Kollision um ein Leben, bis man 87 Leben hat, die man auch nicht mehr verliert. Um Arkanoid durchzuspielen, muß man sich schon um die 2 Stunden Zeit nehmen. Viel Spaß!

Bruce Lee

Hier kann man ganz einfach 99 Leben bekommen: Gehen Sie zu den Tellern, bei denen man normalerweise 1 Leben bekommt. Berühren Sie diese, und Sie haben 99 Leben zur Verfügung. Dieser Trick funktioniert jedoch nur im 2-Player-Modus.

Deflektor

Bei Deflektor gibt es einen recht einfachen Trick. Drücken Sie einfach die <+>- oder <->-Tasten, um in die verschiedenen Level zu kommen.

Great Giana Sisters

Bei diesem süßen Hüpfspiel kommt man einen Level weiter, indem man den Namen des Programmieres eingibt und die Tasten gedrückt hält

Legions of Death

Kaufen Sie Schiffe während Schlachten, denn dann wird Ihnen kein Geld dafür abgeknöpft!

Nebulus

Auch hier gibt es einen Trainermodus. Aktivieren Sie den Pausemodus. Drücken Sie nun die Tasten <Hochpfeil>, <Linkspfeil> und den Buchstaben <J> gleichzeitig. Sie haben nun unendlich viel Zeit. Außerdem kann man noch durch Drücken der Zahlen 1-8 die verschiedenen Level anwählen.

Nemesis

Hier gibt es vier Cheat-Levels. Den ersten erreichen Sie, indem Sie im zweiten Level die Steinschlaufe, die sich am Ende des Levels befindet, aufschießen und sie dann betreten. Der zweite Cheat-Level ist im dritten Level zu erreichen. Dort gibt es zwei Figurenpaare, die Rücken an Rücken stehen. Zerstören Sie beim zweiten Paar die erste Figur, und Sie gelangen wieder in einen Cheat-Modus. Der dritte ist dadurch zu erreichen, daß Sie im siebten Level kurz vor dem Ende in die Öffnung des Ovals fliegen. Den vierten Cheat-Modus erreichen Sie durch Drücken der Taste <Shift Lock>. Viel Spaß bei der zusätzlichen Punktejagd!

Das ist jedoch nicht das einzige, was ich zu Nemesis anzubieten habe. Es gibt in diesem Spiel nämlich auch einen Trainermodus. Drücken Sie einfach während des Fluges des Tasten [J], [K] und [L] gleichzeitig.

One man and his droid

Hier die Codes für dieses Spiel: NONE, BUBBLE, COMMODORE, FINDERS, GENETIC, ZAPPED, TIMEWARP, ECTOPLASM, GORGEOUS, SEASIDE, GIZMO, KING KONG, HOLOGRAM, CURRYRICE, COFFEE, CASSETTE, TELESCOPE, COMPUTER, EDACRAEDA, ALICE.

Parallax

Hier die Codes: STACK, JEVEL, PARCH, SALON, GLOBE.

Raid over Moscow

Bei Raid over Moscow kann man die verschiedenen Sequenzen einzeln zum Üben anwählen. Drücken Sie dazu die Taste <Run/Stop>-<Restore> mit <Q> (Raketensilos), <Pfeil links> (Kreml), <1> (Reaktorraum), <2> (Schlußszene). Dieser Trick funktioniert aber nur, wenn die Weltkarte gerade angezeigt wird.

Spindizzy

Auch in diesem Spiel gibt es einen Cheat-Modus! Geben Sie einfach nach dem Laden die drei Buchstaben PAT ein, und Sie haben unbegrenzte Zeit.

The Great Gurianos

Hier gelangt man einen Level weiter, indem man folgende Tasten gleichzeitig drückt: <M>, <Komma>, <Punkt>, <Cursor rechts>, <Cursor links>. Danach sollten Sie noch die <Return>-Taste betätigen (sofern Sie noch einen Finger frei haben).

Zaxxon

Auch hier gibt es einen Trainermodus. Geben Sie im Titelbild einfach das Wort RED ein, und Sie haben nun unendlich viele Raumschiffe zur Verfügung.

2.7.2 Spiele-POKEs

Wie die Überschrift dieses Kapitels schon sagt, werde ich Ihnen hier einige POKEs verraten, die viel bewirken können. Laden Sie - falls nicht anders gesagt - das Spiel, starten Sie es, und führen Sie dann einen RESET aus. Geben Sie danach die POKEs immer in einer Zeile ein, und starten Sie das Spiel wieder durch den angegebenen SYS-Befehl. Die POKEs sind natürlich ohne Gewähr. Hier nun die versprochenen POKEs:

Antiriad

```
POKE 35486,165 : POKE 35496,16 : SYS 2080
```

Arcana

```
POKE 12933,0 : POKE 12934,2 : SYS 4096
```

Breakthru

```
POKE 6604,173 : SYS 2560
```

Druid

```
POKE 39271,255 : SYS 5120
```

Elevator Action

```
For a=50911 to 50915: POKE a,234 : Next : SYS 53200
```

Firetrap

```
POKE 7500,234 : POKE 7501,234 : SYS 4096
```

Galvan

```
POKE 30602,234 : POKE 30603,234 : POKE 30603,234 : SYS 12288
```

Great Giana Sisters

```
POKE 2446,255 : POKE 6697,255 : SYS 2098
```

Gyroscope

```
POKE 46687,76 : POKE 46688,105 : POKE 46689,182 : SYS 2067
```

He-Man

```
POKE 12651,234 : POKE 12652,234 : POKE 12653,234 : POKE 17610
```

Krackout

```
POKE 32934,0-100  (Levelangabe) : SYS 32837
```

Mutants

```
POKE 9273,230 : SYS 4096
```

Nemesis

```
POKE 5975,234 : POKE 5976,234 : POKE 5977,234 : SYS 5779
```

Parallax

```
POKE 5796,96 : SYS 319
```

Ranarama

POKE 37104,96 : POKE 33969,234 : POKE 33970,234 : SYS 32768

Space Harrier

POKE 5834,234 : POKE 5834,234 : POKE 5836,234 : SYS 2128

Terra cognita

POKE 26703,255 : SYS 24576

Warhawk

POKE 27090,234 : POKE 27091,234 : POKE 27092,234 : SYS 24604

Wizball

POKE 37052,0 : POKE 65303,0 : POKE 65356,0 : POKE 65395,0 : RUN

Xevious

POKE 5605,76 : POKE 5606,31 : POKE 5607,X (Leben) : SYS 5000

2.8 Universalmodule

Steckmodule für den Commodore 64 sind in der letzten Zeit nicht mehr allzu groß in Mode, nicht zuletzt wohl der relativ hohen Preise wegen. Eine Sorte von Steckmodulen erfreut sich aber nach wie vor großer Beliebtheit, die sogenannten "Universalmodule".

Universalmodule enthalten eine Sammlung meist mehr oder minder nützlicher Funktionen der unterschiedlichsten Kategorien. Da die Funktionen alle im Modul untergebracht sind, stehen sie jederzeit sofort und zum Teil sogar innerhalb anderer Programme zur Verfügung.

Durch den hardware-mäßigen Aufbau der Module sind auch Effekte möglich, die man nur mit Software allein gar nicht oder nur sehr schwer erreichen könnte. So vefügen einige Module über eine sogenannte Freeze-Funktion zum "Einfrieren" eines Programms. Die Freeze-Funktion ist in zweierlei Hinsicht sehr nützlich.

Jeder Computerspieler wird das kennen: Man steckt gerade mitten in der wildesten Raumschlacht oder befindet sich bei einem Spiel in einem höheren Level, bei dem ein bestimmtes Zeitlimit vorgegeben ist, und da klingelt das Telefon oder man wird sonst irgendwie unterbrochen. Hat man jetzt ein entsprechendes Modul im Expansion-Port eingesteckt, so genügt ein Druck auf den Freeze-Knopf, und das Spiel wird "eingefroren". Durch einen erneuten Druck auf den Freeze-Knopf des Moduls kann man es dann später jederzeit wieder fortsetzen, als ob nichts gewesen wäre.

Der zweite Vorteil des "Freezens": Sobald ein Programm erst einmal "eingefroren" ist, läßt sich der gesamte Speicher des Commodore 64 und damit auch das Programm auf Diskette oder Kassette ablegen. Damit läßt sich jeder noch so raffinierte Kopierschutz auf der Originaldiskette eines Programms leicht überlisten. Voraussetzung dafür ist allerdings, daß das betreffende Programm komplett in den Rechnerspeicher geladen wird.

Das Freezen ist also eine nützliche Hilfe, wenn man sich von seinen wertvollen Originaldisketten eine Sicherheitskopie anfertigen möchte. Potentielle Raubkopierer werden an den Freezern keine große Freude haben. Die "gefreezt" gespeicherten Programme können nämlich nur mit dem Modul im Expansion-Port zum Laufen gebracht werden.

Ebenso nützlich sind die in vielen Modulen eingebauten Floppyspeeder zum Beschleunigen der Floppy 1541. Wer schon häufiger mit der Floppy gearbeitet hat, wird sicher schon festgestellt haben, daß die Datenübertragung von der Floppy zum Commodore 64 und umgekehrt nicht gerade schnell ist. Bei längeren Programmen kann es schon zwei bis drei Minuten dauern, bis das Programm im Rechner zur Verfügung steht. Verglichen mit der Datasette ist das zwar ein recht guter Wert, bekanntlich kann es ja aber nie schnell genug gehen.

Findige Hardware-Bastler haben deshalb bisher eine ganze Reihe sogenannter Floppyspeeder entwickelt, mit denen man seine Floppy "tunen" kann. Der große Nachteil dieser Hardware-Zusätze: sie sind mit zum Teil über 200 Mark sehr teuer und müs-

sen in die Floppy eingebaut werden. Die rein software-mäßig arbeitenden Floppybeschleuniger der Universalmodule sind da schon wesentlich günstiger (man bekommt sie ja mit dem Modul sozusagen kostenlos mit) und erfüllen ebenso ihren Zweck.

Auf der anderen Seite enthalten manche Module einige Funktionen, mit denen sich in der Praxis nur sehr wenig anfangen läßt. Und mit meist um die 100 Mark sind die meisten Module auch nicht gerade billig.

Vor dem Kauf eines Moduls gilt es daher immer abzuwägen, ob die (für einen persönlich) nützlichen oder die unnützen Funktionen überwiegen.

Damit Sie sich einmal einen Überblick über die verschiedenen Fähigkeiten der Universalmodule verschaffen können, möchte ich Ihnen im folgenden drei Module konkret vorstellen: Magic Formel, Final Cartridge III und Action Cartridge+.

Magic Formel

Mit 198 Mark ist Magic Formel relativ teuer, dafür hat das Modul aber auch einiges zu bieten.

Der eingebaute Floppyspeeder beschleunigt das Laden und Speichern von Programmen etwa um den Faktor 15 bis 20. Sequentielle Dateien werden immerhin noch um den Faktor 10 beschleunigt gelesen. Das Formatieren von Disketten reduziert sich auf 11 Sekunden (normalerweise dauert das ja etwa 90 Sekunden). Das eingebaute BACKUP-Programm kopiert eine Diskette in nur knapp einer Minute, eine sehr nützliche Funktion.

Auch an die Datasetten-Besitzer wurde gedacht. Der integrierte Datasetten-Speeder beschleunigt den Lade- und Speichervorgang um das 10-fache.

Wer einen Drucker aus dem PC-Bereich an den Commodore 64 anschließen möchte, wird sich sicher über die integrierte Centronics-Schnittstelle freuen. Ein einfaches User-Port-Centronics-Druckerkabel reicht dann für den Anschluß aus.

Ein weiterer Pluspunkt: Magic Formel belegt im Commodore 64 keinerlei Speicherplatz. Dadurch ergibt sich eine sehr hohe Kompatibilität (Verträglichkeit) zu anderen Programmen. Und wenn es doch einmal Schwierigkeiten geben sollte, kann man Magic Formel auch abschalten.

Magic Formel verfügt über eine GEOS-ähnliche Benutzeroberfläche. Insbesondere die Arbeit mit der Floppy gestaltet sich dadurch sehr einfach. Ähnlich wie im GEOS-Desktop können die meisten Floppy-Funktionen über sogenannte Pulldown-Menüs ausgewählt werden. Aber auch die restlichen Modulfunktionen werden bequem über Menüs aufgerufen.

Das Modul enthält auch ein sehr umfangreiches Malprogramm, das über alle Standardfunktionen verfügt und die Arbeit mit allen 16 Farben des Commodore 64 gestattet.

Nicht nur in diesem Zusammenhang sehr interessant ist auch die sogenannte Hardcopy-Funktion. Magic-Formel ermöglicht es, aus fast allen Programmen heraus eine Hardcopy des aktuellen Bildschirms zu drucken, egal, ob sich der Rechner gerade im Text- oder im Grafikmodus befindet. Unterstützt werden dabei alle gängigen Druckertypen.

Natürlich verfügt Magic Formel auch über die eingangs erwähnte Freeze-Funktion zum Abspeichern des kompletten Rechnerspeichers auf Diskette oder auch Kassette. Die eingebaute BASIC-Erweiterung ergänzt das BASIC 2.0 des Commodore 64 um etwa 50 neue Befehle. Der Schwerpunkt liegt dabei auf den Grafikbefehlen. Diese sind zum Großteil analog zu den Funktionen des Malprogramms aufgebaut.

Für fortgeschrittene Programmierer am interessantesten dürfte das Maschinensprache-Entwicklungspaket sein. Ähnlich wie beim Freezen bietet ein Modul hier einige Vorteile, die sich mit einer reinen Software-Lösung nicht realisieren lassen. Das Maschinensprache-Paket besteht aus einem Monitor und einem sogenannten 2-Paß-Assembler. Der Assembler dient zum Erstellen der Maschinenprogramme, mit dem Monitor kann man sie dann sehr komfortabel austesten.

Der Monitor besteht im wesentlichen aus vier Teilen, einem Speichermonitor, einem Floppy-Monitor sowie einem Sprite- und einem Zeichensatzeditor. Der große Vorteil dabei: Der Monitor läßt sich praktisch von jedem anderen Programm heraus aufrufen. Danach kann man sich einen "Einblick" in alle Speicherzellen des Rechners verschaffen und einzelne Speicherinhalte nach Belieben verändern. Mit einem speziellen Befehl kann man anschließend wieder in das unterbrochene Programm zurückkehren.

Dieses Verfahren ist nicht nur für Maschinensprache-Programmierer von großer Bedeutung. Es ist für jeden nützlich, der an einem Anwendungsprogramm irgendwelche Änderungen vornehmen möchte. Sehr beliebt ist beispielsweise die Manipulation von Spielen. Vorausgesetzt, man kennt die Adressen der entsprechenden Speicherzellen, kann man sich so leicht sein Zeitlimit erhöhen oder sich zusätzliche "Leben" verschaffen. Einige Tips dazu habe ich Ihnen ja im letzten Abschnitt gegeben.

Alles in allem bietet Magic Formel eine große Anzahl sehr interessanter Funktionen. Insbesondere der Floppy- und Datasetten-Speeder, die Centronics-Schnittstelle, die Hardcopy- und die Freeze-Funktion sowie der Maschinensprache-Monitor dürften für jeden Commodore-64-Anwender von Nutzen sein.

Final Cartridge III

Auch das Final Cartridge III verfügt über einen schon fast obligatorischen Floppy-Speeder und eine Freeze-Funktion zum Abspeichern des kompletten Rechnerspeichers.

Bei der Benutzeroberfläche ist man aber einen großen Schritt weitergegangen. Diese bietet zum Teil sogar mehr Komfort als GEOS. So kann man beispielsweise die einzelnen Fenster frei auf dem Bildschirm bewegen und auch miteinander überlappen.

Ähnlich wie bei GEOS befindet man sich nach dem Start des Moduls in einer Art "Desktop", von wo aus man die restlichen Modulfunktionen über Pulldown-Menüs erreichen kann.

An GEOS orientiert sind auch die verschiedenen Hilfsprogramme. So gibt es beispielsweise eine kleine Textverarbeitung namens "Notepad" zum Ablegen von Notizen. Mit Hilfe der "Clock Settings" kann man eine Alarmzeit einstellen. Der eingebaute Taschenrechner "Calculator" beherrscht die vier Grundrechenarten und arbeitet neunstellig.

Eine sehr nützliche Kleinigkeit sind die "Preferences"-Fenster. Dort kann man die verschiedensten Voreinstellungen treffen. So lassen sich beispielsweise die Tastatur-Wiederholgeschwindigkeit, die Bildschirmfarben sowie die Farbe für den Cursor-Zeiger am Bildschirm den eigenen Wünschen anpassen.

Auch an die BASIC- und Maschinensprache-Programmierer wurde gedacht. Eine kleine BASIC-Erweiterung enthält nützliche Programmierhilfen, wie RENUMBER zum Neunumerieren eines BASIC-Programms oder DUMP zur Ausgabe der Variableninhalte sowie einige Peripheriebefehle zur Vereinfachung der Arbeit mit der Floppy und dem Drucker.

Der Maschinensprache-Monitor ist zwar nicht so umfangreich wie der von Magic Formel, bietet aber alle Standardfunktionen und ebenfalls einen Sprite- und einen Zeichensatzeditor. Die Befehle in der Übersicht:

A	Maschinenprogramm eingeben
C	Speicherinhalte vergleichen
D	Maschinenprogramm ausgeben
EC	Zeichensatzeditor
ES	Sprite-Editor
F	Speicherbereich mit einem Wert füllen
G	Maschinenprogramm starten
H	Speicherinhalte suchen
I	Text eingeben
L	Maschinenprogramm laden
M	Speicheranzeige
OD	Diskettenmonitor
P	Druckerausgabe
R	Register-Ausgabe
S	Maschinenprogramm speichern

T	Speicherbereich verschieben
X	Monitor verlassen
#	Dezimal-Hexadezimal-Umwandlung
$	Hexadezimal-Dezimal-Umwandlung
%	Floppy-Befehl senden
*R	Diskettensektor lesen
*W	Diskettensektor schreiben

Ein separater Assembler ist leider nicht eingebaut. Für kleinere Maschinenprogramme genügt aber auch der im Monitor enthaltene Assembler-Befehl.

Ein großer Pluspunkt des Final Cartridge III ist seine Hardcopy-Funktion. Wer sehr häufig Hardcopies des aktuellen Text- oder Grafikbildschirms erstellen muß, ist mit dem Final Cartridge sicher gut bedient.

Farbige Grafiken werden in entsprechende Graustufen umgesetzt. Wahlweise läßt sich der Bildschirm auch invertiert oder seitwärts ausdrucken und bis auf das Neunfache der Normalgröße vergrößern, wobei man die horizontale und vertikale Vergrößerung getrennt einstellen kann.

Je nach gewünschter Qualität des Ausdrucks kann man mit mehreren Druckdichten und Druckgeschwindigkeiten arbeiten. Sogar der Ausdruck auf einem 24-Nadel-Drucker wird unterstützt.

Das Final Cartridge III hat zwar nicht den Leistungsumfang von Magic Formel, dafür kostet es mit etwa 99 Mark aber auch nur die Hälfte.

Action Cartridge+

Ebenfalls für etwa 99 Mark zu haben ist das Action Cartridge+. Im Gegensatz zu den beiden anderen Modulen bietet es keine grafische Benutzeroberfläche, hat dafür aber einige Besonderheiten, die es von den anderen Modulen abheben.

Zum Beispiel die Floppy- und Datasetten-Speeder. Diese sind nicht nur extrem schnell, sondern lassen sich auch vollständig auf eine Diskette bzw. Kassette aufbringen. Um das betreffende Programm dann beschleunigt zu laden, muß das Modul nicht mehr im Expansion-Port eingesteckt sein!

Die Hardcopy-Funktion weist eine Besonderheit auf. Man kann mit ihr nicht nur den aktuellen Bildschirm auf dem Drucker ausgeben, sondern auch auf Diskette abspeichern lassen, um ihn dann beispielsweise mit einem Grafikprogramm weiterzuverarbeiten.

Recht nützlich für Computerspieler ist auch der sogenannte Sprite-Killer. Damit läßt sich die Sprite-Kollisionskontrolle des 64'ers abschalten. Wem der Platz auf seinen Disketten zu knapp wird, der kann sich seine Programme von einer speziellen Pack-Routine komprimieren lassen.

Die weiteren Funktionen orientieren sich am Leistungsumfang anderer Module. Eine kleine BASIC-Erweiterung mit Programmierhilfen und Floppy-Befehlen fehlt ebensowenig wie ein leistungsfähiger Speicher- und Floppy-Monitor sowie eine am User-Port realisierte Centronics-Schnittstelle.

2.9 Sonstiges

In den vorangegangenen Abschnitten habe ich mich bewußt auf solche Anwendungen konzentriert, die von allgemeinem Interesse sind. Vielleicht möchten Sie Ihren Commodore 64 ja aber für etwas ganz spezielles einsetzen und suchen nun nach passender Hard- und Software.

Eine riesige Fundgrube in dieser Hinsicht ist der Anzeigenmarkt von Zeitschriften, insbesondere der der Zeitschrift 64'er, die sich in der Zwischenzeit zur Standard-Zeitschrift für den Commodore 64 entwickelt hat.

Da der Commodore 64 ja mittlerweile schon einige Jahre auf dem Markt ist, ist auch das Spezialangebot entsprechend vielfäl-

tig. Da gibt es zum Beispiel Programme, mit denen man seinen Biorhythmus berechnen kann, Astrologie- und Psychologieprogramme und vieles mehr.

Vor dem Kauf solcher Spezialprogramme sollte man sich aber noch mehr als bei Standard-Software von deren Qualität überzeugen. Oft handelt es sich dabei nämlich um schnell "zusammengestrickte" BASIC-Programme, die Sie sich mit einigermaßen guten Programmierkenntnissen in ein paar Stunden auch selbst programmieren können.

Nicht zu verachten sind auch die in Zeitschriften abgedruckten Programm-Listings. Viele Zeitschriftenlistings können sich dabei durchaus mit kommerziell vertriebenen Progammen messen, was die Qualität und den Leistungsumfang betrifft. Gerade wenn man etwas sehr spezielles sucht, wird man bei den Listings noch am ehesten fündig. Der große Nachteil dabei: Man muß die Listings eben erst einmal abtippen, bevor man die Programme nutzen kann. Manche Zeitschriften bieten aber auch einen speziellen Service an, bei dem man sich alle Programme einer oder mehrerer Ausgaben auf Disketten schicken lassen kann. Solche Disketten kosten dann meist nicht mehr als etwa 20 bis 30 Mark.

Ein Gebiet, in dem sich der Commodore 64 auch sehr gut einsetzen läßt, ist der Großbereich "Messen-Steuern-Regeln". Wer gerne den Lötkolben schwingt und nach einem "intelligenten" Steuergerät für seine diversen Schaltungen sucht, ist mit dem Commodore 64 sicher bestens bedient. Durch seinen sehr flexibel programmierbaren User-Port eignet sich der 64'er sehr gut zur Steuerung von Schaltungen jeder Art. Beispielsweise gibt es als Ergänzung zu den bekannten Fischertechnik-Baukästen spezielle Bausätze, mit Hilfe derer man den Commodore 64 zur Robotersteuerung einsetzen kann.

3. BASIC-Programmierung leichtgemacht

Dieses Kapitel ist für all diejenigen gedacht, die bisher noch über keinerlei Programmiererfahrungen verfügen. Bekanntlich ist ja aller Anfang schwer. Deshalb habe ich versucht, alles so ausführlich wie möglich zu erklären.

Manch einem wird die eine oder andere Erklärung vielleicht zu umfangreich erscheinen. Trotzdem sollte man das Kapitel auf jeden Fall ganz durcharbeiten, denn es bietet einen umfassenden Einstieg in die BASIC-Programmierung auf dem Commodore 64.

Auch wer schon die ersten Programmierexperimente hinter sich hat, wird sicher noch vieles finden, was ihm bisher unbekannt war.

3.1 Die ersten Schritte

In Kapitel 1 habe ich Ihnen bereits kurz die grundlegende Arbeit mit der Tastatur erklärt und Ihnen auch gezeigt, wie man den Editor des BASIC 2.0 zum Ändern und Ergänzen eingegebener Texte nutzen kann. Wiederholen wir das ganze einmal:

Umgang mit Tastatur und Editor

Nachdem Sie den Commodore 64 eingeschaltet haben und der sogenannte Cursor am Bildschirm erschienen ist, ist der Rechner zur Programmierung in BASIC bereit.

Mit dem Cursor können Sie an jede Stelle des Bildschirms fahren. Dazu stehen Ihnen die beiden Cursor-Steuertasten rechts unten auf der Tastatur zur Verfügung.

Der Cursor markiert immer die Position, an der ein Zeichen, dessen Taste Sie drücken, am Bildschirm erscheint.

Sie können mit dem Cursor problemlos über vorhandene Zeichen fahren. Wenn Sie sich aber in der letzten Bildschirmzeile befinden und die <Cursor down>-Taste betätigen, wird der gesamte Bildschirminhalt um eine Zeile nach oben geschoben (diesen Vorgang bezeichnet man übrigens als "Scrollen" des Bildschirms). Die oberste Bildschirmzeile geht dabei verloren. Das sollten Sie immer beachten!

Falls der Cursor auf einem Zeichen steht und Sie eine Taste drücken, so wird das vorhandene Zeichen durch das zu der gedrückten Taste gehörende Zeichen überschrieben. Auf diese Weise kann man einfache Tippfehler leicht korrigieren.

Oft hat man aber ein oder mehrere Zeichen zuviel oder zuwenig eingetippt. Was macht man dann? Für diesen Fall gibt es die <Inst/Del>-Taste. Wenn Sie diese Taste allein drücken, wird das Zeichen das links vom Cursor steht gelöscht und der Rest der Bildschirmzeile um eine Position nach links verschoben.

Wenn Sie die Taste zusammen mit einer der beiden <Shift>-Tasten drücken, wird der Inhalt der Bildschirmzeile um eine Position nach rechts geschoben, und an der Cursor-Position entsteht eine Lücke, in die Sie ein Zeichen einfügen können.

Probieren Sie die Taste einmal anhand des READY-Schriftzugs auf dem Bildschirm aus. Fahren Sie dazu auf das "D" von READY, und betätigen Sie die <Inst/Del>-Taste.

Jetzt dürfte nur noch REDY in der Bildschirmzeile stehen, und der Cursor steht weiterhin auf dem "D". Er ist mit dem "D" um eine Bildschirmposition nach links gewandert.

Drücken Sie nun die <Inst/Del>-Taste zusammen mit einer <Shift>-Taste. Zwischen dem "E" und dem "D" wird eine Leerstelle eingefügt, auf der der Cursor steht. In diese Lücke können Sie nun wieder das "A" oder jedes andere Zeichen einfügen.

Mit der <Inst/Del>-Taste lassen sich natürlich auch mehrere Zeichen löschen. Wenn Sie die Taste längere Zeit gedrückt hal-

ten, wandert der Cursor von rechts nach links Zeile für Zeile nach oben und löscht alles, was ihm in den Weg kommt.

Um den gesamten Bildschirm zu löschen, wäre dieses Verfahren etwas mühsam. Deshalb gibt es dazu eine spezielle Taste, die mit <Clr/Home> bezeichnet ist.

Wenn Sie diese Taste allein drücken, wandert nur der Cursor in die linke obere Ecke, in seine Home-Position. Nur, wenn Sie die Taste zusammen mit einer <Shift>-Taste betätigen, wird der gesamte Bildschirminhalt gelöscht.

Der Bildschirminhalt geht dabei unwiederbringlich verloren! Man sollte die Löschfunktion daher nur mit Bedacht anwenden. Den Teil des Commodore 64, der den Bildschirm verwaltet und uns diese komfortablen Funktionen zur Verfügung stellt, bezeichnet man übrigens als "Editor". Der Editor ist sozusagen eine speziell für die Programmentwicklung entworfene Textverarbeitung. Aus diesem Grund weist er im Vergleich zu "normalen" Textverarbeitungen einige Besonderheiten auf.

Wie Sie vielleicht schon bemerkt haben, funktioniert beispielsweise die Löschfunktion nicht über den ganzen Bildschirm korrekt. Wenn Sie die Taste dauerhaft gedrückt halten, werden zwar kontinuierlich Zeichen gelöscht, irgendwann (spätestens nach zwei Bildschirmzeilen) verliert der Cursor aber seinen "Anhang", also die Zeichen, die er normalerweise mitschleppt. Diese bleiben einfach am Anfang einer Bildschirmzeile stehen und der Cursor wandert eine Zeile höher.

Der Editor arbeitet also zeilenorientiert, genauer gesagt 2-Zeilen-orientiert. Warum gerade zwei Zeilen?

Nun, wie Sie später noch genauer sehen werden, darf eine einzelne Zeile eines BASIC-Programms bis zu achtzig Zeichen lang sein. Eine Bildschirmzeile ist aber nur vierzig Zeichen lang. Um aber dennoch eine komplette BASIC-Zeile ändern zu können, werden jeweils zwei Bildschirmzeilen sozusagen zu einer Zeile zusammengefaßt. Bedingt durch diesen Sachverhalt ergibt sich ein interessanter Effekt.

Löschen Sie dazu einmal den Bildschirm und schreiben dann in die erste Bildschirmzeile den folgenden Text:

```
DIESES IST DIE ERSTE BILDSCHIRMZEILE
```

Anschließend bringen Sie den Cursor in die Home-Position, indem Sie die <Clr/Home>-Taste betätigen, und fahren dann mit der <Cursor down>-Taste in die zweite Bildschirmzeile, in die Sie diesen Text schreiben:

```
DIESES IST DIE ZWEITE BILDSCHIRMZEILE
```

Jetzt drücken Sie noch einmal die <Clr/Home>-Taste. Nun wollen wir am Anfang der ersten Bildschirmzeile Platz für neue Zeichen schaffen. Halten Sie dazu die <Shift>+<Inst/Del>-Taste dauerhaft gedrückt. Sobald das "E" am Ende von BILDSCHIRMZEILE die letzte Zeilenposition erreicht bzw. überschritten wird, wird die zweite Bildschirmzeile um eine Zeile nach unten geschoben und der Schriftzug der ersten Zeile wandert in die zweite!

Das ganze funktioniert allerdings nur so lange, bis der Schriftzug am Ende der zweiten Bildschirmzeile angekommen ist. Danach können Sie die <Shift>+<Inst/Del>-Taste so lange drücken, wie Sie wollen. Der Schriftzug wandert um keine Stelle nach rechts weiter.

Wo wir gerade schon dabei sind, uns mit den Besonderheiten des Editors vertraut zu machen, kann ich Sie auch gleich noch auf einen anderen Effekt hinweisen.

Versuchen Sie nun doch einmal, den Cursor mit Hilfe der Cursor-Steuertasten zu bewegen. Es geht nicht! Egal welche Steuertaste Sie drücken, der Cursor wandert immer um eine Stelle nach rechts, und an der momentanen Cursor-Position erscheinen merkwürdig aussehende Zeichen. Erst, wenn Sie die Insert-

Strecke überwunden haben, sich also auf dem "D" von DIESES befinden, funktioniert die Cursor-Steuerung wieder richtig.

Durch das Betätigen der <Shift>+<Inst/Del>-Taste sind Sie in einen speziellen Modus gelangt, in dem die Cursor-Bewegungen nicht mehr ausgeführt, sondern mit Hilfe der am Bildschirm erscheinenden sogenannten Steuerzeichen sozusagen "gespeichert" werden.

In unserem Fall ist dieser Effekt eher ärgerlich als hilfreich. Wir werden aber später noch Anwendungen kennenlernen, bei denen man ihn nutzbringend einsetzen kann.

Damit haben Sie wesentliche Eigenschaften und Tücken des Editors bereits kennengelernt. Es wird Zeit, daß wir unser erstes BASIC-Programm schreiben.

Die ersten Programmierversuche

Auf dem Bildschirm herumzuschreiben ist zwar recht interessant, zu irgendeiner Reaktion können sie den Commodore 64 damit aber nicht bewegen. Wie Sie vielleicht noch wissen, müssen Sie die <Return>-Taste drücken, um eine Eingabe an den Rechner weiterzuleiten.

Sobald Sie in einer Bildschirmzeile die <Return>-Taste betätigen, holt sich der Commodore 64 die komplette Zeile in einen internen Speicherbereich und analysiert sie.

Wenn Sie die <Return>-Taste jetzt drücken, bekommen Sie höchstwahrscheinlich entweder gar keine Reaktion (der Cursor stand dann in einer leeren Bildschirmzeile) oder nur die Meldung SYNTAX ERROR, zu gut deutsch also "Syntaxfehler" (der Cursor stand in diesem Fall in einer Bildschirmzeile mit irgendeinem undefinierten Inhalt).

Um mit dem Commodore 64 zu kommunizieren, ihm also Anweisungen zu geben, müssen sie recht strenge Regeln beachten. Der Commodore 64 hat nämlich nur einen eng begrenzten

"Wortschatz" und reagiert auch auf grammatikalische Fehler, ein vergessenes Komma etwa, sehr empfindlich.

Im allgemeinen muß eine Anweisung an den Rechner 100-prozentig korrekt sein, damit der Commodore 64 sie akzeptiert. Es gibt zwar einige Ausnahmen, auf die ich aber nicht eingehen möchte. Es ist besser, wenn Sie sich von vorneherein an die jeweils korrekte Schreibweise gewöhnen.

Eine Anweisung, die der Commodore 64 beispielsweise versteht ist PRINT. PRINT, zu deutsch "drucke", veranlaßt den Rechner, etwas auf dem Bildschirm auszugeben, genauer gesagt das, was hinter dem PRINT-Befehl steht. Probieren wir es gleich einmal aus:

```
PRINT 2*3
```

Bitte vergessen Sie nicht, die <Return>-Taste zu betätigen. Nachdem Sie das getan haben, schreibt der Rechner eine "6" auf den Bildschirm. PRINT kann aber noch wesentlich mehr, zum Beispiel auch Texte ausgeben:

```
PRINT "HALLO"
```

Wie Sie sehen, muß der Text in Anführungszeichen stehen. Lassen Sie die Anführungszeichen einmal weg, schreiben also:

```
PRINT HALLO
```

Was erhalten Sie jetzt auf dem Bildschirm? Eine Null! Der Commodore 64 faßt das HALLO als sogenannte Variable, genauer gesagt als "numerische Variable" auf. Diesen Begriff kennen Sie sicher aus dem Mathematikunterricht. Für eine Variable in einer Formel darf man beliebige Zahlen einsetzen. Nehmen wir ein Beispiel:

```
D=A+B+C
```

In dieser Formel haben wir die vier Variablen A, B, C und D. Setzt man A=1, B=2 und C=3, so wird D der Wert 6 zugewiesen. Genauso dürfen Sie das auch beim Commodore 64 angeben:

```
A=1
B=2
C=3
D=A+B+C
PRINT D
```

Bitte drücken Sie in jeder Zeile die <Return>-Taste! Der PRINT-Befehl gibt wieder eine "6" aus.

Die aktuellen Werte der Variablen werden vom Commodore 64 in einem speziellen Speicher, dem sogenannten Variablenspeicher, abgelegt, von wo sie jederzeit abgerufen werden können.

Der Rechner kann aber nicht nur Zahlen speichern, sondern auch Texte. Damit der Rechner weiß, daß eine Variable einen Text enthalten soll, müssen Sie an ihren Namen das Dollarzeichen [$] anhängen, und der Text muß, wie schon beim PRINT-Befehl, in Anführungszeichen stehen.

```
T$="HALLO"
```

beispielsweise weist der Textvariablen T$ den Text "HALLO" zu. Mit

```
PRINT T$
```

bringen Sie den Text wieder auf den Bildschirm. Anstelle der Bezeichnung "Text" hat sich übrigens allgemein die Bezeichnung "String" eingebürgert, T$ ist dann also eine "String-Variable".

Die Anweisungen, die Sie dem Commodore 64 bisher gegeben haben, hat er immer sofort ausgeführt. Wir haben im sogenannten "Direktmodus" gearbeitet. Größere Programmieraufgaben lassen sich im Direktmodus aber nur schlecht bewältigen. Sobald die ganze Bildschirmseite vollgeschrieben ist, müßte man ja den Bildschirm zumindest teilweise löschen, um neue Anweisungen eingeben zu können. Aus diesem Grund hat man die Möglichkeit, Anweisungen an den Rechner in einzelnen Programmzeilen abzulegen. Mehrere Programmzeilen zusammen bilden ein Programm, genauer gesagt ein BASIC-Programm.

Die Programmzeilen werden, ähnlich wie die Werte der Variablen, in einem internen Speicher des Commodore 64, dem Programmspeicher, abgelegt. Damit der Rechner Programmzeilen von Direkteingaben unterscheiden kann, muß man ihnen eine Nummer voranstellen. Dabei sind Nummern von 0 bis 65.535 erlaubt.

Die Nummern dienen auch der Organisation der Programmzeilen. Die Programmzeilen werden nämlich in aufsteigender Reihenfolge ihrer Nummern gespeichert. Eine Programmzeile mit der Nummer 10 liegt also vor einer mit der Nummer 20 usw...

Das erste Programm

Schreiben wir also unser erstes Programm! Ich habe ein Beispiel gewählt, mit dessen Funktion wohl jeder zurecht kommen wird.

Stellen Sie sich vor, Sie müßten (vielleicht als Handwerker) eine Rechnung schreiben. Für die einzelnen Leistungen verlangen Sie einen bestimmten Preis. Der Einfachheit halber wollen wir einmal annehmen, daß es genau drei Leistungen sind.

Auf jeden Preis muß die gesetzliche Mehrwertsteuer von 14 Prozent aufgeschlagen werden. Außerdem muß der Gesamtpreis berechnet werden, wobei die darin enthaltene Mehrwertsteuer extra ausgewiesen wird. Etwas weiter unten ist das komplette Programm zu dieser Aufgabe abgedruckt. Bitte sehen Sie es sich nur einmal an, tippen es aber noch nicht ab!

Ein kleiner Tip am Rande: Nicht nur, wenn man mit einem Fernseher arbeitet, ist es im allgemeinen besser, beim Programmieren auf die Groß-/Kleinschrift umzuschalten. Die Zeichen sind dann wesentlich besser lesbar. Längere Programmstücke sind im folgenden deshalb in Kleinschrift abgedruckt.

Wie man die Schrift umschaltet, wissen Sie ja sicher noch: Einfach die <Commodore>- und die <Shift>-Taste gleichzeitig drücken.

```
100 print "rechnung erstellen"
110 print "————————————————"
120 input "preis 1: ";p1
130 input "preis 2: ";p2
140 input "preis 3: ";p3
150 m1=p1*0.14
160 m2=p2*0.14
170 m3=p3*0.14
180 mw=m1+m2+m3
190 gp=p1+p2+p3
200 ig=gp+mw
210 print "————————————————"
220 print "preis 1: ";p1;" mehrwertsteuer: ";m1
230 print "preis 2: ";p2;" mehrwertsteuer: ";m2
240 print "preis 3: ";p3;" mehrwertsteuer: ";m3
250 print "————————————————"
260 print "gesamtpreis: ";gp
270 print "mehrwertsteuer:";mw
280 print "insgesamt:";ig
290 end
```

Sie könnten nun einfach daran gehen, das ganze Listing (damit meint man ein ausgedrucktes Programm) Zeichen für Zeichen abzutippen und am Ende jeder Zeile jeweils die <Return>-Taste drücken. Das führt sicher zum Ziel, ist aber viel zu umständlich!

Wenn Sie sich die einzelnen Programmzeilen anschauen, werden Sie feststellen, daß sich manche Zeilen nur in zwei oder drei Zeichen unterscheiden. Hier kann man sich viel Tipparbeit ersparen.

Geben wir das Programm einmal zusammen ein: Bitte tippen Sie zunächst die Programmzeilen 100 bis 120 ein.

Die Zeilen 130 und 140 sind mit Zeile 120 fast identisch. Fahren Sie deshalb mit dem Cursor wieder auf die Zeile 120, und überschreiben Sie die 2 in 120 durch eine 3. Dann fahren Sie mit dem Cursor in der Zeile weiter und ändern den "preis 1" in "preis 2" sowie die Variable "p1" in "p2" um. Danach drücken Sie wieder die <Return>-Taste. So haben Sie aus der Zeile 120 eine Zeile 130 gemacht.

Um die Zeile 140 einzugeben, verfahren Sie analog. Also einfach die Ziffern wieder um eins erhöhen und zum Schluß nicht vergessen, die <Return>-Taste zu drücken.

Die Zeilen 150 bis 170 sehen wieder sehr ähnlich aus. Tippen sie also zuerst die Zeile 150 komplett ab und erzeugen daraus dann die Zeilen 160 und 170.

Geben Sie nun die Zeilen 180 bis 200 ein.

Die Zeile 210 ist mit Zeile 110 völlig identisch. Fahren Sie also mit dem Cursor nach oben und ändern die 110 in 210 um. Anschließend drücken Sie die <Return>-Taste.

Auch die Zeile 250 ist mit der Zeile 110 identisch. Ändern Sie die 210 also noch schnell in 250 um und drücken noch einmal die <Return>-Taste.

Die Zeilen 220 bis 240 sind bis auf drei Zeichen identisch. Wie Sie diese drei Zeilen am schnellsten in den Rechner bekommen, müßten Sie jetzt selbst wissen. Zum Schluß tippen Sie noch die Zeilen 260 bis 290 ab.

Geschafft! Bitte löschen Sie nun einmal den Bildschirm und geben dann ein:

```
LIST (+<Return>)
```

Das komplette Programm-Listing erscheint nun auf dem Bildschirm. LIST kann aber noch mehr:

Manchmal möchten Sie vielleicht nur einen Teil eines Programms auf dem Bildschirm haben. Insbesondere dann, wenn Sie ein größeres Programm haben, das nicht ganz auf den Bildschirm paßt. Zu diesem Zweck können Sie hinter LIST die Nummern der Zeilen angeben, die Sie gelistet haben wollen. Dabei gibt es vier Möglichkeiten:

```
LIST 200
```

listet nur die Programmzeile 200.

```
LIST 100-200
```

bringt die Zeilen 100 bis 200 auf den Bildschirm.

```
LIST -200
```

listet alle Programmzeilen vom Programmanfang bis einschließlich Zeile 200.

```
LIST 200-
```

gibt alle Programmzeilen ab Zeile 200 bis zum Programmende aus. Um die Feinheiten des Editors abzuhaken, möchte ich Ihnen auch gleich noch erklären, wie Sie eine Programmzeile im Programmspeicher löschen können. Dazu geben Sie einfach nur die Nummer der zu löschenden Zeile ein. Möchten Sie beispielsweise die Zeile 210 löschen, so geben Sie ein:

```
210 (+<Return>)
```

Das gesamte im Rechnerspeicher befindliche Programm löschen Sie durch Eingabe von:

```
NEW (+<Return>)
```

Bitte geben Sie den Befehl NEW aber keinesfalls jetzt ein, und gehen Sie auch später immer sehr sorgfältig damit um! Ist das Programm erst einmal gelöscht, kann man es (zumindest mit normalen Mitteln) nicht mehr zurückholen. Sicher brennen Sie nun darauf, das Programm ablaufen zu lassen. Nichts einfacher als das! Bitte geben sie ein:

```
RUN (+<Return>)
```

Der Commodore 64 schreibt daraufhin die Überschrift

```
RECHNUNG ERSTELLEN
```

auf den Bildschirm und fragt Sie mit

```
PREIS 1: ?
```

nach dem ersten Preis. Hinter dem Fragezeichen blinkt der Cursor. Bitte tippen Sie nun irgendeine Zahl ein, beispielsweise 1000, und drücken die <Return>-Taste. Und schon möchte der Rechner mit

```
PREIS 2: ?
```

den nächsten Preis wissen. Bitte geben Sie wieder eine Zahl ein und drücken <Return>. Dasselbe wiederholt sich noch einmal für den dritten Preis.

Danach führt der Rechner die Berechnungen in den Programmzeilen 150 bis 200 aus. Das geht so schnell, daß Sie es zeitlich gar nicht bemerken. Der Commodore 64 beherrscht übrigens alle vier Grundrechenarten:

```
Additon:        +
Subtraktion:    -
Multiplikation: *
Division:       /
```

Ab Programmzeile 210 werden dann mit Hilfe des PRINT-Befehls die Ergebnisse ausgegeben. Der END-Befehl in Zeile 290 bildet den Abschluß jedes BASIC-Programms. Er darf auch irgendwo mitten in einem Programm stehen. Trifft der Rechner auf ein END, dann beendet er die Programmabarbeitung und kehrt in den Direktmodus zurück.

Das Programm steht danach im Rechnerspeicher weiterhin zur Verfügung und kann mit RUN erneut gestartet oder mit LIST gelistet werden.

Wenn, wie bei unserem Programm, das "logische" Ende des Programms (also die Stelle, an der die Programmabarbeitung beendet wird) mit dem tatsächlichen Programmende zusammenfällt (was meistens der Fall ist), dann darf man die END-Anweisung auch weglassen.

Ich hoffe, daß das Programm bei Ihnen korrekt gelaufen ist. Falls Sie irgendwann einen SYNTAX ERROR IN ... bekommen haben sollten, dann listen Sie bitte die betreffende Zeile und

schauen nach, ob Sie sich nicht vertippt haben. Nachdem Sie den Fehler korrigiert haben (<Return>-Taste drücken nicht vergessen!), starten Sie das Programm erneut mit RUN.

Auf die beiden Befehle INPUT und PRINT werde ich gleich noch ausführlicher eingehen. Zunächst sollten Sie das Programm dauerhaft sichern. Wie Sie ja bereits wissen, geht das Programm im Rechnerspeicher nach dem Ausschalten des Rechners unwiederbringlich verloren.

Programme dauerhaft sichern

Bevor Sie ein Programm auf einer neuen Diskette speichern können, müssen Sie diese zunächst formatieren. Falls Sie nicht mehr wissen, wie das geht, hier noch einmal die entsprechende Anweisung:

```
OPEN 1,8,15,"N:DISKETTENNAME,ID":CLOSE 1 (+<Return>)
```

Den Diskettennamen können Sie frei wählen. Wie wäre es zum Beispiel mit "BASIC-DISK"? Als ID geben Sie eine zweistellige Kombination aus Buchstaben und/oder Ziffern an. Am besten wird es sein, wenn Sie sich Ihre Disketten von "01" bis "99" durchnumerieren. Es ist sehr wichtig, daß jede Ihrer Disketten über eine andere ID verfügt! Die Floppy unterscheidet die einzelnen Disketten, die Sie während der Arbeit am Rechner in das Laufwerk schieben, nämlich anhand dieser ID. Sobald der Formatiervorgang beendet ist, können Sie unser Rechnungsprogramm mit

```
SAVE "RECHNUNG",8 (+<Return>)
```

auf der Diskette ablegen. Wenn Sie es später wieder laden wollen, geben Sie einfach ein:

```
LOAD "RECHNUNG",8 (+<Return>)
```

Bei der 8 hinter dem Programmnamen handelt es sich übrigens um die sogenannte "Geräteadresse" der Floppy. Die Geräteadresse signalisiert dem Rechner, welches Peripheriegerät er ansprechen soll. Datasetten-Besitzer geben anstelle der 8 eine 1 an.

Auf die Feinheiten der Floppy-Programmierung möchte ich an sich erst in Kapitel 5 eingehen. Sozusagen als kleinen Vorgeschmack darauf erkläre ich Ihnen hier schon ein paar Kleinigkeiten:

Jede Diskette hat ein Inhaltsverzeichnis, auch Directory genannt, in das die Namen sämtlicher Programme eingetragen werden, die Sie auf der Diskette speichern. Wie kommt man nun an dieses Inhaltsverzeichnis heran?

Das Inhaltsverzeichnis hat einen speziellen Namen, das Dollarzeichen [$], unter dem es wie jedes andere "Programm" in den Rechnerspeicher geholt werden kann.

Achtung: Wenn Sie es noch nicht getan haben, dann speichern Sie das Rechnungsprogramm bitte unbedingt jetzt auf die Diskette. Durch das Einladen des Directories geht das Programm (im Rechnerspeicher) verloren! Um das Directory zu laden, geben Sie ein:

```
LOAD "$",8 (+<Return>)
```

und anschließend:

```
LIST (+<Return>)
```

Danach müßte in etwa folgendes auf dem Bildschirm erscheinen:

```
0 "BASIC-DISK        " 01 2A
2 "RECHNUNG"           PRG
662 BLOCKS FREE.
```

In der ersten Zeile sehen Sie (invertiert dargestellt) den Namen der Diskette. Darunter steht der Name unseres Programms. Die anderen Zahlen und Texte sollen uns im Augenblick nicht interessieren. Mehr dazu in Kapitel 5.

Wenn Sie an einem Programm eine Änderung vornehmen und anschließend diese neue Programmversion unter demselben Na-

men auf der Diskette abspeichern möchten, geht das natürlich nicht. Der Name dient ja gerade zur Unterscheidung der einzelnen Programme auf der Diskette zwei Programme mit demselben Namen ergäben keinen Sinn. Jetzt haben Sie zwei Möglichkeiten: Entweder Sie löschen die alte Programmversion. Dazu geben Sie ein:

```
OPEN 1,8,15,"S:RECHNUNG":CLOSE 1 (+<Return>)
```

Oder Sie geben der alten Programmversion einen anderen Namen, beispielsweise "RECHNUNG(ALT)". Dazu tippen Sie ein:

```
OPEN 1,8,15,"R:RECHNUNG(ALT)=RECHNUNG":CLOSE 1 (+<Return>)
```

Programme ausdrucken

Eine weitere Möglichkeit, seine Programme in einen dauerhaften "Zustand" zu überführen, ist, sie auf dem Drucker auszugeben. Besonders nützlich ist das Ausdrucken bei längeren Programmen. Man bekommt auf dem Papier einfach einen besseren Überblick als bei den jeweils nur 25 Zeilen auf dem Bildschirm. Wie geht man nun konkret vor?

Zuallererst müssen Sie natürlich Ihren Drucker einschalten. (Das vergißt man leicht.)

Ähnlich wie bei der Floppy müssen wir dem Commodore 64 mitteilen, daß wir den Drucker ansprechen möchten. Dazu geben Sie ein:

```
OPEN 4,4 (+<Return>)
```

Nun leiten wir sämtliche Bildschirmausgaben auf den Drucker um:

```
CMD 4 (+<Return>)
```

Das READY, das ja nach jeder Befehlsausführung auf dem Bildschirm erscheint, wird jetzt bereits auf dem Drucker ausgegeben. Jetzt geben Sie einfach ein:

```
LIST (+<Return>)
```

und der Drucker beginnt zu arbeiten. Um anschließend wieder normal auf dem Bildschirm arbeiten zu können, geben Sie zum Schluß noch ein:

```
PRINT#4 (+<Return>)
CLOSE 4 (+<Return>)
```

Das ganze noch einmal im Überblick:

```
OPEN 4,4
CMD 4
LIST
PRINT#4
CLOSE 4
```

Der PRINT-Befehl

Im Grunde genommen besteht jedes Programm vereinfacht betrachtet aus nur drei Teilen:

- dem Eingabeteil, in dem irgendwelche Daten erfragt werden
- dem Verarbeitungsteil, in dem diese Daten irgendwie bearbeitet werden
- dem Ausgabeteil, in dem die bearbeiteten Daten wieder ausgegeben werden

Die Ein- und Ausgabe von Daten bildet also einen wesentlichen Bestandteil jedes Programms. Betrachten wir zunächst einmal die Ausgabe, also den PRINT-Befehl. Etwas haben Sie über PRINT ja schon kennengelernt:

```
PRINT 12*5/3
```

beispielsweise gibt das Rechenergebnis 20 aus.

```
PRINT VR
```

gibt den Inhalt der Variablen VR aus. Texte müssen in Anführungszeichen stehen:

```
PRINT "HALLO"
```

Oft möchte man mehrere Texte hintereinander ausgeben. Da könnte man natürlich einfach schreiben:

```
PRINT "DAS ERGEBNIS IST: ":PRINT EG
```

Es geht aber auch wesentlich einfacher:

```
PRINT "DAS ERGEBNIS IST:";EG
```

Der Strichpunkt ; trennt also zwei Teile einer Ausgabe. Mit Hilfe des Strichpunktes lassen sich beliebige Ausgaben aneinander reihen:

```
PRINT "DAS ERGEBNIS IST: ";12*5/3;" SOWIE: ";EG
```

Der Strichpunkt hat noch eine weitere nützliche Funktion, wenn er ganz am Ende einer PRINT-Anweisung steht. Normalerweise wird bei jeder neuen PRINT-Anweisung die Ausgabe in einer neuen Zeile begonnen. Wenn Sie zum Beispiel schreiben:

```
10 print "ergebnis: "
20 print 20
```

Dann erhalten Sie die folgende Ausgabe:

```
ERGEBNIS:
20
```

Besser wäre es, die 20 würde hinter dem Doppelpunkt stehen. Dazu hängen Sie an die PRINT-Anweisung in Zeile 10 einfach ein [;] an:

```
10 print "ergebnis: ";
20 print 20
```

Der Strichpunkt sorgt also dafür, daß der Schreib-Cursor nicht in die nächste Bildschirmzeile wandert, sondern an seiner momentanen Position stehenbleibt.

Der INPUT-Befehl

Den INPUT-Befehl haben Sie in dem Rechnungsprogramm weiter oben ja auch schon kurz kennengelernt. Er ist grundsätzlich so aufgebaut:

```
10 INPUT "Kommentartext";Eingabevariable
```

Den Kommentartext dürfen Sie auch weglassen. INPUT druckt dann nur sein Fragezeichen aus, um zu signalisieren, daß es von Ihnen etwas wissen will.

```
10 INPUT EG
```

beispielsweise weist die Eingabe der Variablen EG zu. Auch Texte lassen sich eingeben. Dazu müssen Sie aber eine String-Variable als Eingabevariable angeben:

```
10 INPUT EG$
```

String-Eingaben an eine numerische Variable weist der INPUT-Befehl zurück. Ähnlich, wie man bei PRINT mehrere Ausgaben kombinieren kann, so kann man bei INPUT mehrere Eingaben zu einer INPUT-Anweisung zusammenfassen:

```
10 INPUT "BITTE 3 WERTE EINGEBEN: ";W1,W2,W3
```

Die einzelnen Variablen müssen, durch Kommas getrennt, angehängt werden. Um die drei Werte jetzt einzugeben, gibt es zwei Möglichkeiten:

Entweder Sie drücken nach jedem Wert die <Return>-Taste. Der INPUT-Befehl macht sich in diesem Fall (nach dem ersten und dem zweiten Wert) mit einem "??" bemerkbar. Am Bildschirm sieht das so aus:

```
BITTE 3 WERTE EINGEBEN:? 123
?? 321
?? 231
```

Die andere Möglichkeit besteht darin, die einzelnen Werte durch ein Komma zu trennen:

```
BITTE 3 WERTE EINGEBEN:? 123,321,231
```

Dann müssen Sie nur einmal die <Return>-Taste drücken. Falls Sie mehr Werte eingeben, als Variablen hinter INPUT stehen (beim letzten INPUT beispielsweise vier), reagiert der Rechner mit einem EXTRA IGNORED. Das Programm wird aber trotzdem fortgesetzt.

Übrigens, vielleicht haben Sie es selbst schon festgestellt, im Direktmodus ist INPUT nicht erlaubt. Man kann es nur innerhalb von Programmen einsetzen.

Über die Lesbarkeit von Programmen

Wenn Sie später einmal ein Programm, das Sie vor längerer Zeit geschrieben haben, abändern müssen, ist es sehr wichtig, daß Sie überhaupt noch feststellen können, was das Programm wo macht und welche Bedeutung einzelne Programmanweisungen haben.

Nun gut, bei einer INPUT- oder einer PRINT-Anweisung weiß man in der Regel recht schnell, was diese jeweils zu bedeuten haben. Nicht alle BASIC-Befehle, die Sie noch kennenlernen werden, sind aber so leicht durchschaubar wie INPUT und PRINT.

Deshalb sollte man in jedes Programm "Bemerkungen" aufnehmen, die darüber Auskunft geben, was die einzelnen Programmanweisungen zu bedeuten haben. Zu diesem Zweck stellt BASIC einen speziellen Befehl zur Verfügung: REM. REM ist die Abkürzung für "Remark", zu deutsch also "Bemerkung".

Steht in einer Programmzeile ein REM-Befehl, so wird der Rest der Zeile vom Commodore 64 ignoriert. Daher können Sie hinter ein REM beliebige Kommentare schreiben, beispielsweise:

```
10 rem das ist mein erster kommentar
```

Im Gegensatz zu PRINT oder INPUT muß der Kommentartext auch nicht in Anführungszeichen geschrieben werden. Sehen wir uns einmal an einem praktischen Beispiel an, was Kommentare im Hinblick auf die Lesbarkeit eines Programms bewirken können. Nehmen wir dazu unser Rechnungsprogramm:

```
90 rem programm: rechnung
100 print "rechnung erstellen"
110 print "——————————————"
115 rem preise eingeben
120 input "preis 1: ";p1
130 input "preis 2: ";p2
140 input "preis 3: ";p3
145 rem mehrwertsteuer berechnen
150 m1=p1*0.14
160 m2=p2*0.14
170 m3=p3*0.14
180 mw=m1+m2+m3
185 rem gesamtpreis berechnen
190 gp=p1+p2+p3
200 ig=gp+mw
205 rem ergebnisse ausgeben
210 print "——————————————"
220 print "preis 1: ";p1;"  mehrwertsteuer: ";m1
230 print "preis 2: ";p2;"  mehrwertsteuer: ";m2
240 print "preis 3: ";p3;"  mehrwertsteuer: ";m3
250 print "——————————————"
260 print "gesamtpreis: ";gp
270 print "mehrwertsteuer:";mw
280 print "insgesamt:";ig
290 end
```

So sieht das ganze doch schon wesentlich übersichtlicher aus. Optimal ist es aber noch lange nicht. Die einzelnen Programmzeilen sind noch zu sehr "zusammengedrängt". Um das zu ändern, setzt man sogenannte "Leerzeilen" mit einem Doppelpunkt ein. Zunächst zum Doppelpunkt : selbst.

Wie Sie ja inzwischen wissen, darf eine Programmzeile bis zu achtzig Zeichen lang sein. Ein einzelner Befehl wird allerdings kaum je so lang sein. Daher kann man in eine Programmzeile auch mehrere Anweisungen schreiben, die dann aber durch einen Doppelpunkt voneinander getrennt sein müssen.

Die Zeilen 120 bis 140 des Programms könnte man beispielsweise zu einer Zeile zusammenfassen:

```
120 input"preis 1: ";p1:input"preis 2: ";p2:input"preis 3: ";p3
```

Wie Sie selbst sehen, erhöht sich dadurch die Lesbarkeit des Programms aber nicht gerade. Nach Möglichkeit sollte man deshalb in eine Programmzeile nur eine Anweisung schreiben.

Wenn allerdings der Speicherplatz knapp wird (was bei größeren Programmen leider sehr schnell passiert), kommt man oft nicht umhin, mehrere Zeilen zusammenzufassen. Doch zurück zu den Leerzeilen. Eine Leerzeile erzeugt man dadurch, daß man in die Zeile nur einen Doppelpunkt schreibt, beispielsweise:

```
10 :
```

Ändern wir das Programm-Listing einmal entsprechend ab:

```
90 rem programm: rechnung
92 :
94 :
98 :
100 print "rechnung erstellen"
110 print "------------------"
112 :
113 :
115 rem preise eingeben
117 :
120 input "preis 1: ";p1
130 input "preis 2: ";p2
140 input "preis 3: ";p3
142 :
143 :
145 rem mehrwertsteuer berechnen
147 :
150 m1=p1*0.14
160 m2=p2*0.14
170 m3=p3*0.14
180 mw=m1+m2+m3
182 :
183 :
185 rem gesamtpreis berechnen
187 :
190 gp=p1+p2+p3
200 ig=gp+mw
202 :
203 :
205 rem ergebnisse ausgeben
207 :
210 print "------------------"
```

```
220 print "preis 1: ";p1;" mehrwertsteuer: ";m1
230 print "preis 2: ";p2;" mehrwertsteuer: ";m2
240 print "preis 3: ";p3;" mehrwertsteuer: ";m3
250 print "————————————"
260 print "gesamtpreis: ";gp
270 print "mehrwertsteuer:";mw
280 print "insgesamt:";ig
282 :
284 :
286 :
290 end
```

Die einzelnen Programmteile sind nun klar voneinander getrennt und das Programm ist optimal lesbar.

Die letzte Möglichkeit, ein Programm "optisch" zu gestalten, habe ich bereits angewandt: Das Einfügen von Leerzeichen zwischen den einzelnen Komponenten einer Anweisung. Der Commodore 64 hat nämlich die nützliche Eigenschaft, daß er diese Leerzeichen (unabhängig davon, wie viele es sind) grundsätzlich überliest. Anstatt

```
10 input"preis 1";p1
```

schreibt man deshalb besser (wie im Programm geschehen):

```
10 input "preis 1";p1
```

Genauso gut könnte man auch schreiben:

```
10 input "preis 1" ; p1
```

3.2 Strukturiert programmieren

Bisher war es uns nicht möglich, die "Richtung" des Programmablaufs in irgendeiner Weise zu beeinflussen. Das Programm wurde immer von der ersten bis zur letzten Programmzeile abgearbeitet.

Bei einfachen Programmen mag das durchaus ausreichen. Bei größeren Programmen steht man aber oft vor dem Problem, daß ein bestimmter Programmteil nur unter einer bestimmten Bedin-

gung abgearbeitet werden soll. Häufig kommt es auch vor, daß ein Programmteil mehrfach hintereinander oder während des Programmablaufs immer wieder abgearbeitet werden muß.

Nehmen wir einmal an, ein bestimmtes Programmstück wird während der Programmausführung 20 mal durchlaufen. Dieses Programmstück nun 20 mal in den Programmtext zu schreiben, würde das Programm nicht nur unverhältnismäßig "aufblähen", sondern auch Änderungen unnötig erschweren (eine Änderung innerhalb des Programmstücks müßten Sie ja dann 20 mal durchführen).

Aus diesem Grund hält BASIC spezielle Steueranweisungen parat, mit denen man gezielt in den Programmablauf eingreifen und einem Programm eine gewisse "Struktur" verpassen kann.

3.2.1 Sprünge und Verzweigungen

Im einfachsten Fall "springt" das Programm zu einer bestimmten Programmzeile. Dazu gibt es den Befehl GOTO.

Der Befehl GOTO

Hinter GOTO müssen Sie die Nummer der Programmzeile, zu der gesprungen werden soll, angeben.

```
GOTO 1000
```

beispielsweise springt zu der Programmzeile mit der Nummer 1000. Dazu muß Ihr Programm natürlich auch eine Programmzeile mit dieser Nummer enthalten. Falls das nicht der Fall sein sollte, meldet sich der Rechner mit einem UNDEF'D STATEMENT ERROR.

Von wo aus Sie zu welcher Stelle in einem Programm springen, ist völlig egal. Auch die Anzahl der Sprünge ist unbegrenzt. Sie können also nach Herzenslust "kreuz und quer" in einem Programm herumspringen.

Damit sollte man allerdings sehr vorsichtig sein. Schneller als man denkt, weiß man nicht mehr so recht, wohin das Programm jeweils eigentlich springt, und vor allem, weshalb es an eine bestimmte Stelle springt. Versuchen Sie einmal das folgende Programm zu analysieren:

```
100 goto 240
110 print "4. sprung"
120 goto 220
130 print "2. sprung"
140 goto 200
150 print "6. sprung"
160 end
200 print "3. sprung"
210 goto 110
220 print "5. sprung"
230 goto 150
240 print "1. sprung"
250 goto 130
```

Zugegeben, das ganze ist reichlich übertrieben. Wenn man aber nicht aufpaßt, hat man sehr schnell eine derartig unübersichtliche Programmstruktur, wie sie im obigen Beispiel demonstriert wurde. Ein guter Programmierer sollte deshalb eigentlich völlig ohne GOTO auskommen!

Wesentlich sinnvoller als ein unbedingter Sprung mit GOTO ist die Verzweigung zu einer Programmzeile in Abhängigkeit von einer bestimmten Bedingung.

Der IF-THEN-Befehl

Dafür ist der IF-THEN-Befehl zuständig. Der Name "WENN-DANN" sagt schon fast alles. Wenn eine bestimmte Bedingung erfüllt ist, dann wird etwas getan, beispielsweise zu einer Programmzeile gesprungen. Die genaue Syntax sieht wie folgt aus:

```
IF 'Bedingung' THEN 'Anweisungen, falls Bedingung wahr'
```

IF prüft also, ob die angegebene Bedingung wahr ist. Falls ja, werden die hinter THEN stehenden Anweisungen ausgeführt. Ist die Bedingung falsch, dann wird in der nächsten Programmzeile weitergemacht.

```
IF VR=1 THEN PRINT "VR ENTHAELT DEN WERT EINS"
```

beispielsweise gibt den hinter THEN stehenden Text nur aus, wenn VR den Wert eins enthält. Die Bedingungen hinter IF können sehr vielfältig sein:

```
IF WT<5 THEN .....
```

prüft, ob WT kleiner als 5 ist.

```
IF WT>10 THEN .....
```

führt die hinter THEN stehenden Anweisungen nur aus, falls WT einen Wert größer als 10 enthält. Textvariablen lassen sich ebenfalls prüfen:

```
IF TX$="ENDE" THEN END
```

beendet den Programmablauf, falls TX$ die Zeichenkette "ENDE" enthält. Auch sogenannte logische Operatoren dürfen verwendet werden:

```
IF A>1 AND A<10 THEN .....
```

prüft, ob der Wert der Variablen A zwischen 1 und 10 liegt. Sehr häufig verwendet man IF-THEN, um in Abhängigkeit von einer Bedingung zu einer anderen Programmzeile zu springen. Dabei gibt es eine Besonderheit in der Schreibweise. Anstelle von

```
IF X=5 THEN GOTO 1000
```

kann man genauso gut

```
IF X=5 THEN 1000
```

oder

```
IF X=5 GOTO 1000
```

schreiben. Ein nicht zu unterschätzender Nachteil der IF-THEN-Anweisung ist die Tatsache, daß die hinter THEN stehenden Anweisungen auf eine Programmzeile beschränkt sind.

In vielen Fällen bräuchte man wesentlich mehr Platz. Hier gibt es aber einen einfachen Trick. Die Prüfbedingung wird einfach negiert, also umgekehrt.

Nehmen wir einmal an, Sie haben die Bedingung "A=10". Falls diese Bedingung zutrifft, soll Programmtext von vielleicht fünf Zeilen Länge abgearbeitet werden. Diese Bedingungsabfrage realisieren Sie wie folgt:

```
100 if a<>10 then 160
110 .....:rem anweisungen,
120 .....:rem falls
130 .....:rem a
140 .....:rem gleich
150 .....:rem 10
160 rem weiterer programmtext
```

Es gibt übrigens einen sehr einfachen Weg, jede Bedingung umzukehren. Dazu müssen wir den logischen Operator NOT bemühen, auf den wir später noch näher eingehen werden. NOT negiert jeden Ausdruck, der hinter ihm in Klammern steht.

```
IF NOT (A<5) THEN .....
```

beispielsweise bedeutet, daß die hinter THEN stehenden Anweisungen abgearbeitet werden sollen, falls A nicht kleiner als 5, also größer oder gleich 5 ist. Manchmal kommt es auch vor, daß mehrere Bedingungen in Abhängigkeit voneinander geprüft werden müssen. Dazu setzt man dann mehrere IF-THEN-Befehle hintereinander.

```
IF A>1 THEN IF B<10 THEN ....
```

prüft zunächst, ob der Wert von A größer als 1 ist, und (falls ja) anschließend, ob B einen Wert kleiner als 10 enthält.

Der ON-GOTO-Befehl

Manchmal unterscheiden sich mehrere Verzweigungsbedingungen nur durch einen konstanten Wert, etwa den Wert 1:

```
IF AW=1 THEN 1000
IF AW=2 THEN 2000
IF AW=3 THEN 3000
IF AW=4 THEN 4000
IF AW=5 THEN 5000
```

In diesem Fall fünf IF-THEN-Anweisungen untereinander zu schreiben, ist recht umständlich. Aus diesem Grund gibt es den ON-GOTO-Befehl:

```
ON AW GOTO 1000,2000,3000,4000,5000
```

ON-GOTO verzweigt in Abhängigkeit von der Variablen AW in die hinter GOTO angegebenen Programmzeilen. Enthält AW den Wert 1, wird zur Zeile 1000 verzweigt, bei AW=2 zur Zeile 2000 usw... Enthält AW einen Wert kleiner als 1 oder größer als 5, dann wird das Programm in der nächsten Programmzeile fortgesetzt.

Nicht immer wird AW so ideale Werte wie in diesem Beispiel enthalten. Unterscheiden sich die Werte beispielsweise um den Wert 5, so kann man sich mit einer einfachen Umrechnung behelfen:

```
ON AW/5 GOTO .....
```

Anstelle einer einfachen Variablen dürfen zwischen ON und GOTO also auch Formeln stehen. Die Anzahl der hinter GOTO stehenden Zeilennummern ist nur durch die Länge einer Programmzeile beschränkt. Einen ganz typischen Anwendungsfall für eine ON-GOTO-Konstruktion zeigt das folgende Listing. Nehmen wir einmal an, Sie haben ein Programm geschrieben, daß fünf Funktionen zur Verfügung stellt, etwa:

```
Daten eingeben
Berechnungen durchführen
Rechnung erstellen
Rechnung ändern
Rechnung ausdrucken
```

Diese Programmfunktionen stellen Sie dem Anwender Ihres Programms in Form eines Auswahlmenüs (ähnlich der Speisekarte in einem Restaurant) zur Verfügung. Aus diesem Menü kann er durch Angabe einer Kennziffer die Funktion auswählen, die er benötigt. Durch die Auswertung dieser Kennziffer mittels ON-GOTO verzweigt das Programm dann an die entsprechende Programmstelle:

```
100 print "auswahlmenue:"
110 print "1: menuepunkt 1"
120 print "2: menuepunkt 2"
130 print "3: menuepunkt 3"
140 print "4: menuepunkt 4"
150 print "5: menuepunkt 5"
160 input "bitte entsprechenden wert (1-5) eingeben: ";aw
170 :
180 on aw goto 1000,2000,3000,4000,5000
190 end
200:
1000 rem programmteil fuer menuepunkt 1
.....
2000 rem programmteil fuer menuepunkt 2
.....
3000 rem programmteil fuer menuepunkt 3
.....
4000 rem programmteil fuer menuepunkt 4
.....
5000 rem programmteil fuer menuepunkt 5
.....
```

3.2.2 Schleifen

Eine Konstruktion, die man in allen Programmen häufig benötigt, sind sogenannte Programmschleifen. Der Programmcode innerhalb einer Schleife wird jeweils mehrfach durchlaufen.

Nehmen wir zum Beispiel nun einmal an, Sie möchten den Text "HALLO" zwanzigmal auf den Bildschirm schreiben.

Da wäre es natürlich ein Unding, dazu auch ein Programm mit zwanzig PRINT-Befehlen zu schreiben.

Die FOR-TO-NEXT-Schleife

Viel einfacher geht es mit einer "Zählschleife". Bei dieser genügt ein einziger PRINT-Befehl:

```
100 for z=11 to 20
110 print "hallo"
120 next z
```

Wie wird diese Schleife nun genau abgearbeitet? In Zeile 100 wird hinter dem FOR durch die Zuweisung "Z=1" eine Zählvariable Z mit dem Startwert 1 definiert. Die 20 hinter dem TO besagt, daß (von der Zahl 1) bis zur Zahl 20 gezählt werden soll.

Der Commodore 64 arbeitet die Zeile 100 ab und merkt sich die definierten Werte in einem speziellen Zwischenspeicher. Anschließend wird die Programmzeile 110 wie gewohnt ausgeführt.

In der Zeile 120 trifft der Rechner auf den NEXT-Befehl. Dieser veranlaßt ihn, den Wert der Variablen Z um eins zu erhöhen. Anschließend vergleicht er den Wert von Z mit dem Endwert 20.

Z hat im Augenblick den Wert 2, ist also kleiner als 20. Daher springt der Commodore 64 auf die hinter dem FOR-TO-Befehl stehende Anweisung (in diesem Fall PRINT) zurück und arbeitet diese noch einmal ab.

Danach kommt wieder der NEXT-Befehl an die Reihe. Z wird wieder um 1 erhöht und hat dann den Wert 3, ist also immer noch kleiner als 20. Die PRINT-Anweisung wird deshalb ein drittes Mal ausgeführt.

So geht das ganze weiter, bis Z einen Wert größer als 20 erreicht hat, was nach dem zwanzigsten Schleifendurchlauf der Fall ist.

Die allgemeine Form der FOR-TO-NEXT-Schleife sieht so aus:

```
FOR Z=Anfangswert TO Endwert
... Schleifenanweisungen ...
NEXT Z
```

Anstelle von Z dürfen Sie natürlich einen beliebigen Variablennamen verwenden. Manchmal kommt einem die Zählschrittweite von 1 etwas ungelegen, man möchte vielleicht in Zweier- oder Dreier-Schritten oder auch rückwärts zählen.

In diesem Fall muß man die gewünschte Schrittweite hinter TO, vom Endwert abgetrennt durch ein STEP, angeben:

```
FOR Z=Anfangswert TO Endwert STEP Schrittweite
... Schleifenanweisungen ...
NEXT Z
100 for z=0 to 50 step 5
110 print "hallo"
120 next z
```

In dieser Schleife wird in Fünfer-Schritten von 0 bis 50 gezählt. Auch Dezimalzahlen sind erlaubt:

```
100 for z=1.5 to 7.75 step 0.25
110 print z
120 next z
```

Noch ein Beispiel mit einer negativen Schrittweite:

```
100 for z=20 to 1 step -1
110 print "hallo"
120 next z
```

Diese Schleife bewirkt dasselbe wie die erste Schleife, nur wird jetzt eben von 20 auf 1 rückwärts gezählt. Fehlt die STEP-Anweisung, so wird, wie wir gesehen haben, immer die Schrittweite 1 genommen.

Übrigens dürfen zwischen der FOR- und der NEXT-Zeile beliebig viele Programmzeilen mit Schleifenanweisungen stehen.

Ebenfalls erlaubt sind weitere FOR-TO-NEXT-Konstruktionen. Man spricht dann von einer "Schachtelung" von Schleifen. Eine Schachtelung mit drei Schleifen könnte zum Beispiel so aussehen:

```
100 for z1=1 to 1000
110 :
115 :
120 :
125 for z2=1 to 100
130 :
135 :
140 for z3=1 to 10
150 :
160 rem schleifenanweisungen
170 :
180 next z3
190 :
200 :
210 next z2
220 :
230 :
240 :
250 next z1
```

Die äußere Z1-Schleife erstreckt sich von Zeile 100 bis 250. Eine Ebene tiefer folgt die Z2-Schleife von Zeile 125 bis 210 und dann schließlich die Z3-Schleife von Zeile 140 bis 180. Sehr wichtig ist, daß die Schleifen auch wirklich verschachtelt sind, eine innere Schleife also komplett innerhalb der äußeren Schleife liegt. Eine Konstruktion, wie

```
125 for z2=1 to 100
130 :
135 :
140 for z3=1 to 10
150 :
160 rem schleifenanweisungen
170 :
180 next z2
190 :
200 :
210 next z3
```

ist nicht erlaubt! Die beiden Schleifen sind hier ja nicht verschachtelt, sondern überlagern sich, denn die Z1-Schleife erstreckt sich von Zeile 125 bis 180 und die Z3-Schleife von Zeile 140 bis 210. Noch eine Anmerkung zu NEXT: Den Namen der

Schleifenvariablen hinter NEXT kann man weglassen. In der Regel ergeben sich dadurch keine Komplikationen. Der Commodore 64 nimmt, falls der Name fehlt, immer die Variable der zuletzt abgearbeiteten FOR-TO-Anweisung. Die Konstruktion

```
100 for z=1 to 20
110 print "hallo"
120 next
```

ist also völlig korrekt. Um Platz zu sparen, kann man das ganze natürlich auch in eine Zeile "quetschen":

```
100 for z=1 to 20:print "hallo":next
```

Eine weitere Besonderheit ergibt sich bei verschachtelten Schleifen. Hier darf man mehrere NEXT-Anweisungen zu einer zusammenfassen. Das Beispiel von weiter oben, ließe sich verkürzt so schreiben:

```
100 for z1=1 to 1000
125 for z2=1 to 100
140 for z3=1 to 10
150 :
160 rem schleifenanweisungen
170 :
180 next z3,z2,z1
```

Besser lesbar wird das ganze dadurch allerdings nicht gerade. Man sollte sich daher immer genau überlegen, ob die "kompakte" Schreibweise wirklich nötig ist. Bei vielen Programmierproblemen, die eine Schleife erfordern, hat man keinen Zähler, den man für eine FOR-TO-NEXT-Schleife verwenden könnte. Vielmehr hat man irgendeine Abbruchbedingung (beispielsweise "Anwender drückt eine Taste"), die meist innerhalb der Schleife irgendwann auf "wahr" gesetzt wird.

Die Frage ist nun, wo prüft man die Abbruchbedingung? Am Anfang der Schleife, irgendwo in der Mitte oder am Ende der Schleife. Daraus ergeben sich drei Schleifenkonstruktionen:

```
WHILE-DO (Während-Tue)
EXITIF (Verzweige, falls)
REPEAT-UNTIL (Wiederhole-Bis)
```

Leider sind diese Schleifenkonstruktionen im BASIC 2.0 des Commodore 64 nicht direkt verfügbar. Man kann sie sich aber mit GOTO und IF-THEN relativ leicht selbst "basteln".

Die While-Do-Schleife

Bei der While-Do-Schleife wird die Abbruchbedingung ganz am Anfang der Schleife getestet. Nehmen wir einmal an, die Bedingung wäre "A=10", d.h., so lange die Variable den Wert 10 enthält, soll die Schleife durchlaufen werden.

```
100 rem while-do
110 :
120 if not (A=10) then 200
130 .....:rem schlei-
140 .....:rem fen-
150 .....:rem anwei-
160 .....:rem sun-
170 .....:rem gen
180 goto 120
190 :
200 rem weiterer programmtext
```

Die Schleife erstreckt sich von Zeile 120 bis 180. Da die IF-Anweisung ja innerhalb einer Zeile stehen muß, negieren wir die Abbruchbedingung einfach mit NOT und überspringen die Schleifenanweisungen, falls die Bedingung nicht zutrifft.

Eine While-Do-Schleife hat den großen Vorteil, daß sie, falls die Abbruchbedingung von vorneherein "wahr" ist, kein einziges Mal durchlaufen wird. Eine FOR-TO-NEXT-Schleife dagegen wird immer mindestens einmal abgearbeitet.

Die Exitif-Schleife

Bei der Exitif-Schleife wird die Abbruchbedingung irgendwo innerhalb der Schleifenanweisungen getestet. Bleiben wir einmal bei der Bedingung "A=10":

```
100 rem exitif
110 :
120 .....:rem schleifen-
130 .....:rem anwei-
140 .....:rem sungen
```

```
150 if a=10 then 200:rem aussprung
160 .....:rem schleifen-
170 .....:rem anwei-
180 .....:rem sungen
185 goto 120
190 :
200 rem weiterer programmtext
```

Die Exitif-Schleife ist die flexibelste von allen. Nicht zuletzt, weil innerhalb der Schleife natürlich beliebig viele Abbruchtests stehen dürfen.

Die Repeat-Until-Schleife

Die Repeat-Until-Schleife arbeitet die Schleifenanweisungen so lange ab, bis die Abbruchbedingung eingetreten ist. Die Bedingung wird deshalb am Ende der Schleife kontrolliert.

Bleiben wir bei unserer Abbruchbedingung "A=10". Bei Repeat-Until heißt das, die Schleife soll so lange durchlaufen werden, bis A den Wert 10 enthält. Deshalb müssen wir uns wieder mit einer Negation der Bedingung in der IF-Anweisung behelfen:

```
100 rem repeat-until
110 :
120 .....:rem schlei-
130 .....:rem fen-
140 .....:rem anwei-
150 .....:rem sun-
160 .....:rem gen
170 if not (A=10) then 120
190 :
200 rem weiterer programmtext
```

Welche Schleifenart sich jeweils am besten eignet, ergibt sich meist unmittelbar aus dem jeweiligen Programmierproblem. Manchmal werden Sie vielleicht gar nicht gleich merken, daß Sie eigentlich eine Schleife programmiert haben.

Trotzdem wollte ich Ihnen hier einmal kurz die grundsätzlichen Schleifentypen vorstellen. Das wird Ihnen später nicht zuletzt bei der Analyse fremder Programme (dadurch erweitert man seine Programmierfähigkeiten immer noch am besten!) hilfreich sein.

3.2.3 Unterprogramme

Oft soll ein bestimmter Programmteil in einem Programm zwar mehrfach abgearbeitet werden, aber nicht direkt hintereinander. Eine Programmschleife scheidet dann natürlich aus.

Eine Konstruktion mit GOTO-Befehlen läßt sich ebenfalls nicht realisieren. Zwar könnte das Programm zu dem bertreffenden Programmteil hinspringen. Nachdem der Programmteil abgearbeitet ist, wüßte der C64 nicht, wohin er zurückspringen soll.

Der Commodore 64 müßte sich also irgendwie "merken" können, von wo aus der Programmteil aufgerufen wurde. Und das kann er auch!

Der GOSUB-RETURN-Befehl

Dazu müssen Sie nur anstelle des GOTO den Befehl GOSUB verwenden. An das Ende des aufzurufenden Programmteils hängen Sie ein RETURN an, und fertig ist das Unterprogramm:

```
100 gosub 1000:rem erster aufruf
110 ...
120 gosub 1000:rem zweiter aufruf
130 ...
140 gosub 1000:rem dritter aufruf
.....
200 end:rem ende des hauptprogramms
900 :
910 :
920 :
1000 rem unterprogramm
.....
.....
.....
2000 return:rem ruecksprung
```

Die Programmzeilen 100 bis 200 stellen in diesem Programm das Hauptprogramm dar. In den Zeilen 1000 bis 2000 steht ein Unterprogramm, das vom Hauptprogramm aus dreimal aufgerufen wird. Beim Aufruf des Unterprogramms durch GOSUB 1000 passiert folgendes:

Der Rechner merkt sich in einem internen Zwischenspeicher die Position der Aufrufstelle im Programm (beim ersten Aufruf die Zeile 100). Anschließend fährt er ab der Zeile 1000 mit der Programmabarbeitung fort.

Sobald er auf den RETURN-Befehl in Zeile 2000 trifft, holt er sich aus dem Zwischenspeicher die Position der Aufrufstelle und setzt dann genau dahinter (bzw. in der nächsten Zeile) die Programmabarbeitung fort. Eine an sich recht einfache, aber ungeheuer effiziente Technik!

Die Bezeichnungen "Hauptprogramm" und "Unterprogramm" muß man relativ sehen. Von einem Unterprogramm aus kann man nämlich jederzeit ein anderes Unterprogramm aufrufen. Sogar der Selbstaufruf von Unterprogrammen ist möglich.

```
100 gosub 1000 ──────────┐
110 .....                │
120 .....                │
130 .....                │
200 end                  │
900 :                    │
910 :                    │
1000 rem unterprogramm 1 <┘
1010 .....
1020 gosub 2000 ────────┐
1030 .....              │
.....                   │
1500 return             │
1900 :                  │
1910 :                  │
2000 rem unterprogramm 2 <┘
2010 .....
2020 .....
2030 .....
.....
2500 return
```

Das von Zeile 100 aus aufgerufene Unterprogramm 1 ruft seinerseits das Unterprogramm 2 auf, das sich von Zeile 2000 bis Zeile 2500 erstreckt. Man spricht in diesem Fall von einer Schachtelung von Unterprogrammen. Trifft der Rechner in einem Programm auf ein RETURN, ohne zuvor ein GOSUB abgearbeitet zu haben, so reagiert er mit der Fehlermeldung RETURN WITHOUT GOSUB ERROR.

Diese Fehlermeldung bekommt man schneller als man denkt! Vergißt man nämlich am Ende des Hauptprogramms die END-Anweisung, so verzweigt der Rechner, nachdem er das Hauptprogramm abgearbeitet hat, zwangsläufig in das erste Unterprogramm. Sobald er dann auf dessen RETURN-Befehl trifft, erscheint die Fehlermeldung.

Der ON-GOSUB-Befehl

Analog zu GOTO gibt es auch bei GOSUB die Möglichkeit einer "Sammelverzweigung". Dazu steht der ON-GOSUB-Befehl zur Verfügung:

```
ON WT GOSUB 1000,2000,3000
```

Bei WT=1 verzweigt das Programm nach Zeile 1000, bei WT=2 nach Zeile 2000 usw... Nachdem das Unterprogramm abgearbeitet ist, wird das Programm hinter dem ON-GOSUB-Befehl bzw. in der nächsten Programmzeile fortgesetzt. Mit Hilfe der Unterprogrammtechnik und ON-GOSUB lassen sich Programme jeder Größenordnung sehr übersichtlich und flexibel gestalten.

Häufig benötigte Programmteile, etwa eine spezielle Eingaberoutine oder eine komfortable Ausgaberoutine, sollte man immer in Form von Unterprogrammen schreiben und dann als separates Programm auf Diskette ablegen. So erhält man nach und nach eine größere Unterprogrammsammlung.

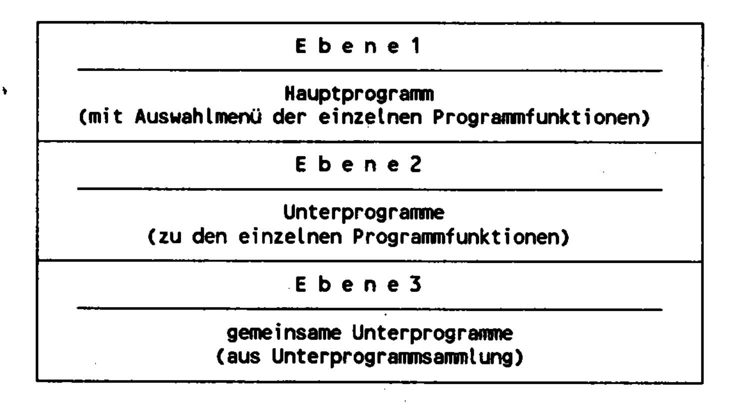
Möchte man dann später ein neues Programm schreiben, sucht man sich einfach die passenden Unterprogramme heraus und bindet sie in das Programm ein. Die Programmentwicklungszeit wird dadurch wesentlich verkürzt. Wie man zwei Programme "zusammenbindet" (dieser Vorgang wird oft auch als "mergen" bezeichnet), erkläre ich Ihnen übrigens in Kapitel 5 (für die Floppy) bzw. in Kapitel 10 (für die Datasette).

Bei der Beschreibung von ON-GOTO weiter oben habe ich Ihnen schon gezeigt, wie man ein Programm durch Zerlegung in einzelne Teile entsprechend den Programmfunktionen sehr über-

sichtlich aufbauen kann. Wenn wir das ganze etwas abwandeln, können wir mit Unterprogrammen arbeiten:

```
100 print "auswahlmenue:"
110 print "1: menuepunkt 1"
120 print "2: menuepunkt 2"
130 print "3: menuepunkt 3"
140 print "4: menuepunkt 4"
150 print "5: menuepunkt 5"
160 input "bitte entsprechenden wert (1-5) eingeben: ";aw
170 :
180 on aw gosub 1000,2000,3000,4000,5000
190 goto 100
200:
1000 rem unterprogramm fuer menuepunkt 1
.....
1900 return
2000 rem unterprogramm fuer menuepunkt 2
.....
2900 return
3000 rem unterprogramm fuer menuepunkt 3
.....
3900 return
4000 rem unterprogramm fuer menuepunkt 4
.....
4900 return
5000 rem unterprogramm fuer menuepunkt 5
.....
5900 return
```

Grafisch veranschaulicht sieht der Programmaufbau dann so aus:



Die Ebene 3 wird nicht in jedem Programm enthalten sein (in dem obigen Listing ebenfalls nicht). Insbesondere in kleinen Programmen reichen die Unterprogramme zu den einzelnen Programmfunktionen in der Regel aus.

Wenn Sie sich an diesen grundsätzlichen Programmaufbau halten, werden Sie für Ihre Programme nur einen Bruchteil der Zeit benötigen, die Sie aufbringen müßten, wenn Sie einfach "drauflos" programmieren. Ein weiterer Vorteil dieser Technik ist die leichte Erweiterbarkeit des Programms.

Nehmen wir einmal an, Ihnen fällt, nachdem das Programm an sich schon fertig ist, eine weitere nützliche Funktion ein, die Sie unbedingt in das Programm aufnehmen möchten. Wie müssen Sie dann vorgehen? Zunächst schreiben Sie die neue Programmfunktion als Unterprogramm, am besten ab Programmzeile 6000:

```
6000 rem unterprogramm fuer menuepunkt 6
.....
6900 return
```

Anschließend erweitern Sie das Auswahlmenü im Hauptprogramm, um den zusätzlichen Menüpunkt:

```
155 print "6: menuepunkt 6"
160 input "bitte entsprechenden wert (1-6) eingeben: ";aw
```

Zum Schluß erweitern Sie noch die ON-GOSUB-Anweisung um den zusätzlichen Aufruf:

```
180 on aw gosub 1000,2000,3000,4000,5000,6000
```

Das ist schon alles! An den anderen Programmfunktionen müssen Sie keinerlei Änderungen vornehmen.

3.3 Daten verwalten, aber wie?

Selbst in einfachsten Programmen haben Sie irgendwelche Daten, die verarbeitet werden müssen. Dabei stellt sich zunächst das Problem, wie die Daten verwaltet werden sollen. Wie und wo soll

man die Daten ablegen? Und wie stellt man sie dann später zur Weiterverarbeitung wieder bereit?

Zunächst einmal muß man zwei grundsätzliche Arten von Daten unterscheiden: Numerische Daten, beispielsweise:

```
12345
459.34
-71
```

und String-Daten, beispielsweise:

```
"ZEICHENKETTE"
"12345"
"DAS IST EIN LANGER TEXT MIT 38 ZEICHEN"
```

Das englische Wort "String" steht für "Zeichenkette". Ein String ist also eine Aneinanderreihung einzelner Zeichen, eben eine Zeichenkette.

Im Gegensatz zu den numerischen Werten wird ein String Zeichen für Zeichen im Rechnerspeicher abgelegt. Jedes Zeichen wird dabei durch einen bestimmten Code, den sogenannten ASCII-Code, repräsentiert. Der Buchstabe "A" beispielsweise hat den ASCII-Code 65, der Buchstabe "B" den Code 66, der Buchstabe "C" den Code 67 usw... Der String

```
"ABC"
```

wird im Rechnerspeicher also durch die Codefolge

```
65 66 67
```

dargestellt. Eine vollständige Tabelle der ASCII-Codes finden Sie übrigens im Anhang. Die maximale Länge eines Strings beträgt 255 Zeichen. Wenn Sie sich jetzt daran erinnern, daß ja eine einzelne Programmzeile maximal 80 Zeichen lang sein darf, dann fragen Sie sich vielleicht, wie man überhaupt je zu einem so langen String kommen soll.

Das geht sehr leicht! Der Commodore 64 erlaubt es nämlich, mit Strings fast so komfortabel wie mit numerischen Werten umzugehen. So kann man z.B. mit + zwei Strings "addieren". Aus

```
"COMMODORE"+" 64"
```

wird

```
"COMMODORE 64"
```

Außerdem stehen eine Reihe spezieller String-Funktionen zur Verfügung, mit denen man aus einem String beliebige Teile "herausschneiden" kann. Mehr dazu etwas weiter unten.

Es gibt auch einen String mit der Länge Null, den sogenannten "Null-String", der als

```
""
```

geschrieben wird. Wie Sie ja in der Zwischenzeit schon zur Genüge gesehen haben, müssen Strings grundsätzlich in Anführungszeichen stehen. Das ermöglicht es dem Rechner, Strings von numerischen Daten zu unterscheiden. Die beiden Schreibweisen

```
12345
```

und

```
"12345"
```

stehen also trotz ihrer Ähnlichkeit für zwei grundverschiedene Datenarten! Im ersten Fall handelt es sich um eine Zahl, mit der man rechnen kann, im zweiten Fall um einen Text, der eben nur aus Ziffern besteht.

3.3.1 Konstanten und Variablen

Wenn man den Wert eines Datums genau kennt und sich dieser Wert während des Programmablaufs auch nicht ändert, schreibt

man ihn am besten als sogenannte "Konstante" ins Programm, für Rechnungen lohnt sich beispielsweise

```
PRINT "MEHRWERTSTEUERSATZ: "
;14; "PROZENT"
```

14 ist in diesem Fall eine Konstante, ebenso "MEHRWERT-STEUERSATZ: " und "PROZENT".

In den meisten Anwendungsfällen kennt man die Werte der einzelnen Daten nicht so genau, oder die Werte können sich während der Programmabarbeitung ändern. Da ist es besser, man verwendet eine sogenannte "Variable".

Das gilt unter Umständen auch für das obige Beispiel. Nehmen wir einmal an, Sie schreiben sich ein Programm zur Berechnung Ihrer Steuern, in dem Sie vielleicht ein Dutzend mal (oder mehr) den Mehrwertsteuersatz für Berechnungen benötigen. Falls sich der Mehrwertsteuersatz nun ändert, müßten Sie ihn im Programmtext an jeder Stelle ändern. Das Kunststück dabei ist dann, diese Stellen überhaupt zu finden.

Wesentlich einfacher wird das ganze, wenn Sie sich eine Variable

```
MEHRWERTSTEUER=14
```

definieren. Was passiert bei dieser Zuweisung? Der Commodore 64 vermerkt den Namen der Variablen sowie ihren Wert in seinem internen Variablenspeicher. Beim Namen gibt es allerdings eine wichtige Einschränkung: Es werden nur die ersten beiden Zeichen berücksichtigt! Von dem Wort MEHRWERTSTEUER merkt sich der Rechner also nur "ME". Eine Variable

```
MEHRKOSTEN
```

wäre für ihn daher mit MEHRWERTSTEUER identisch. Man sollte deshalb von vorneherein möglichst nur zweistellige Variablennamen verwenden. Wie wäre es anstelle von MEHRWERT-STEUER beispielsweise mit MW?

Dadurch ersparen Sie sich später auch viel Tipparbeit. Die Lesbarkeit eines Programms erhöht sich dadurch aber nicht gerade. Wer weiß schon auf Anhieb, daß mit MW die Mehrwertsteuer gemeint ist? Daher empfiehlt es sich, zu jedem Programm eine Variablentabelle zu erstellen, in der man die Bedeutungen der einzelnen Variablen vermerkt. Am einfachsten legt man diese Tabelle ganz am Anfang eines Programms in REM-Zeilen ab, beispielsweise:

```
100 rem steuerberechnung
110 :
120 rem variablentabelle
130 mw: mehrwertsteuer
140 ek: einkommensteuer
150 z1,z2,z3: zaehlvariable
......
```

Doch kommen wir darauf zurück, wie der Commodore 64 eine Variable behandelt. Er legt ihren Namen und ihren Wert also im Variablenspeicher ab. Wenn Sie die Variable dann später irgendwo im Programm verwenden, also ihren Namen beispielsweise in eine Formel schreiben, sucht der Rechner die Variable im Variablenspeicher und holt sich ihren Wert, den er dann (im übertragenen Sinne) anstelle des Namens in die Formel einsetzt. Wenn Sie also schreiben

```
PRINT A+B+C
```

holt der Commodore 64 nacheinander die Werte der Variablen A, B und C, addiert sie Werte und gibt das Ergebnis aus. Den Wert einer Variablen können Sie jederzeit ändern. Schreiben Sie beispielsweise:

```
100 wt=1:rem erste wertzuweisung
110 print wt
120 wt=2:rem zweite wertzuweisung
130 print wt
140 wt=3:rem dritte wertzuweisung
150 print wt
```

dann erhalten Sie die Ausgabe

```
1
2
3
```

Der Wert einer Variablen richtet sich also immer nach der letzten Wertzuweisung. Falls Sie einmal vergessen haben sollten, einer Variablen einen Wert zuzuweisen, bevor Sie sie irgendwo im Programm verwenden, so ist das auch nicht weiter schlimm. Der Wert der Variablen wird in diesem Fall vom Rechner automatisch auf Null gesetzt. In den Variablen, mit denen wir bisher gearbeitet haben (A, B, C, MW, WT usw.), können wir beliebige numerische Werte ablegen, beispielsweise:

```
A=3626353
B=0.1259
C=-187
```

Häufig benötigt man aber nur ganze Zahlen aus einem begrenzten Zahlenbereich. Nehmen wir zum Beispiel die Zählschleifen mit FOR-TO-NEXT. Hier kommt man in der Regel mit Zahlen zwischen 0 und 10.000 aus.

Aus diesem Grund gibt es einen zweiten Variablentyp für numerische Werte, die sogenannten "Integer-Variablen". Unter einem "Integer-Wert" versteht man immer einen ganzzahligen Wert, also

```
1, 2, 3, 4 usw. und -1, -2, -3 usw.
```

nicht aber

```
1.5 oder 0.3.
```

Integer-Variablen können daher nur ganzzahlige Werte, genauer gesagt ganzzahlige Werte zwischen -32767 und 32768 aufnehmen. Um dem Commodore 64 mitzuteilen, daß Sie eine Integer-Variable definieren möchten, müssen Sie ihrem Namen das Prozentzeichen % anhängen, beispielsweise:

```
A%=10
B%=-20
CD%=1000
```

Der dritte auf dem Commodore 64 vorhandene Variablentyp dient zur Aufnahme von Strings. Wie eine String-Variable markiert werden muß, wissen Sie ja bereits: Man hängt ihrem Namen das Dollarzeichen [$] an, beispielsweise:

```
TX$="TEXT"
EG$="EINGABE"
W$="9876543210"
```

usw. Fassen wir einmal kurz zusammen. Auf dem Commodore 64 gibt es drei verschiedene Variablentypen:

Numerische Variablen (oft auch Real-Variable genannt), bei denen keine besondere Namenskennung erforderlich ist.

Integer-Variablen, die mit % hinter dem Namen markiert werden müssen.

String-Variablen, die mit $ gekennzeichnet werden müssen.

3.3.2 Felder

Häufig hat man in einem Programm eine größere Anzahl gleichartiger Daten. Nehmen wir einmal an, Sie möchten die Telefonnummern von fünf Freunden in Variablen speichern. Da könnten Sie beispielsweise schreiben:

```
T1$="123456"
T2$="654321"
T3$="132456"
T4$="456123"
T5$="564321"
```

Viel einfacher in der späteren Handhabung wird das ganze aber, wenn Sie die Werte in einem sogenannten "Variablenfeld" ablegen. Dazu schreiben Sie:

```
T$(1)="123456"
T$(2)="654321"
T$(3)="132456"
T$(4)="456123"
T$(5)="564321"
```

Durch diese Schreibweise haben Sie ein Variablenfeld mit dem Namen T definiert. Worin unterscheidet sich nun das T1$ von dem T$(1) bzw. das T2$ von dem T$(2)?

Bei T1$, T2$ usw. handelt es sich um voneinander unabhängige Einzelvariablen. T$(1), T$(2) usw. dagegen sind einzelne Komponenten einer einzigen Variablen bzw. eines Variablenfeldes. Veranschaulichen wir uns das ganze einmal grafisch:

T1	123456

T2	654321

T3	132456

T4	456123

T5	564321

T	1	123456
	2	654321
	3	132456
	4	456123
	5	564321

Der große Vorteil eines Feldes besteht nun in den Zugriffsmöglichkeiten auf die einzelnen Feldkomponenten. Jede Feldkomponente wird einfach durch ihre Nummer in Klammern angesprochen. Diese Nummer bezeichnet man übrigens als "Index" des Feldes. Um beispielsweise die Feldinhalte auf dem Bildschirm auszugeben, schreiben Sie:

```
FOR I=1 TO 5:PRINT T$(I):NEXT I
```

Bei T1$, T2$ müßten Sie jede Variable extra angeben:

```
PRINT T1$;T2$;T3$;T4$;T5$
```

Bei nur fünf Werten ist der Unterschied nicht allzu groß. Stellen Sie sich aber einmal vor, Sie hätten vielleicht 100 Werte! Abgesehen davon, daß Sie schon bei der Namensgebung für die einzelnen Variablen Probleme bekommen würden, bräuchten Sie mehrere Programmzeilen, um alle Werte ausgeben zu lassen.

Sind die Werte in einem Feld W mit 100 Elementen, die mit W(1) bis W(100) angesprochen werden können, abgelegt, so genügt ein:

```
FOR I=1 TO 100:PRINT W(I):NEXT I
```

Wenn Sie das jetzt gleich einmal ausprobieren wollen, werden Sie Probleme bekommen. Sobald Sie versuchen, einem Feldelement mit einem Index größer als 11 einen Wert zuzuweisen, reagiert der Commodore 64 mit einem BAD SUBSCRIPT ERROR.

Das Problem dabei ist die Speicherplatz-Reservierung im Variablenspeicher. Jedes Element eines Feldes benötigt ja eine gewisse Anzahl Speicherplätze.

Deshalb muß man dem Rechner mitteilen, wie groß das Feld maximal werden soll (ob es dann später tatsächlich so groß wird, spielt keine Rolle), man muß das Feld "dimensionieren".

Dazu gibt es einen sehr einfach gebauten BASIC-Befehl: DIM. Hinter DIM wird einfach das größtmögliche Feldelement angegeben:

```
DIM W(100)
```

richtet also ein Feld W mit 100 Elementen ein. Man kann mit einem DIM-Befehl auch mehrere Felder dimensionieren:

```
DIM A(10),B(20),C(30)
```

Das Feld A wird mit 10, das Feld B mit 20 und das Feld C mit 30 Elementen dimensioniert.

Vergißt man die Dimensionierung, so setzt der Rechner die Feldgröße auf 11 Elemente, d.h., bei der ersten Wertzuweisung an ein Element eines Feldes, das bis zu diesem Zeitpunkt noch nicht dimensioniert war, wird für das Feld ein Speicherplatz für 11 Elemente reserviert.

Hinweis: Eine einmal durchgeführte Dimensionierung kann während der Programmabarbeitung nicht mehr abgeändert werden!

Eine Konstruktion wie

```
100 dim w(90):rem 1. dimensionierung
.....
300 dim w(100):rem 2. dimensionierung
.....
```

ist daher nicht erlaubt und führt zu einem REDIM'D ARRAY ERROR. Man muß sich deshalb wirklich von vorneherein darüber klar sein, wie viele Elemente ein Feld aufnehmen soll.

Um beim Beispiel der Telefonnummern zu bleiben: So, wie die Nummern im Augenblick gespeichert sind, müssen Sie wissen, welche Nummer zu welchem Freund gehört. Wesentlich günstiger wäre es, wenn die Namen der Freunde passend zu jeder Telefonnummer gespeichert wären.

Dazu könnte man nun zwei weitere Felder einrichten, etwa NN$(1)-NN$(5) für die Nachnamen und VN$(1)-VN$(5) für die Vornamen. Warum die gesamten Daten aber nicht in einem einzigen Feld zusammenfassen?

Wir bräuchten also ein Feld, das zu jedem Freund die Daten Nachname, Vorname und Telefonnummer aufnimmt. Sehen wir uns das zuerst einmal grafisch an:

		1	2	3
AD	1	Mueller	Klaus	123456
	2	Maier	Heinz	654321
	3	Schulz	Georg	132456
	4	Schmidt	Uwe	456123
	5	Mayr	Peter	564321

Wie bekommen wir das ganze nun in ein Variablenfeld? Wenn Sie sich die Tabelle einmal genauer anschauen, werden Sie feststellen, daß man jedes Element durch zwei "Koordinaten" ansprechen kann. Der Vorname "Uwe" beispielsweise steht in der 4. Zeile und der 2. Spalte, hat also die Koordinaten 4,2.

Was wir also benötigen, ist ein Feld, bei dem man die einzelnen Elemente mit eben diesen "Koordinaten" ansprechen kann:

```
DIM AD$(5,3)
```

Durch diese Anweisung wird ein zweidimensionales Feld AD$ eingerichtet. "Zweidimensional" heißt, daß man zwei Indizes benötigt, um auf ein einzelnes Feldelement zuzugreifen. Ein dreidimensionales Feld würde demnach drei Indizes erfordern, beispielsweise:

```
F(1,2,3)
```

Anschaulich können Sie sich solch ein Feld als eine Art "Würfel" vorstellen (das zweidimensionale Feld in der Abbildung oben dagegen stellt ja eine ebene Fläche dar). Bei vier- oder gar fünf Dimensionen, beispielsweise

```
F(1,2,3,4) oder F(1,2,3,4,5)
```

wird es mit der Veranschaulichung etwas kritisch. Mehr als drei Dimensionen gibt es in unserer räumlichen Welt ja nicht. Ich kann Sie aber beruhigen. Mehr als zwei Dimensionen werden Sie bei der Programmierung kaum je benötigen. Bleiben wir also bei

unserem Beispiel. Um die Daten in das jeweilige Feld zu bekommen, gehen Sie am besten wie folgt vor:

```
10 ad$(1,1)="mueller":ad$(1,2)="klaus":ad$(1,3)="123456"
20 ad$(2,1)="maier":ad$(2,2)="heinz":ad$(2,3)="654321"
30 ad$(3,1)="schulz":ad$(3,2)="georg":ad$(3,3)="132456"
40 ad$(4,1)="schmidt":ad$(4,2)="uwe":ad$(4,3)="456123"
50 ad$(5,1)="mayr":ad$(5,2)="peter":ad$(5,3)="564321"
```

Mit der Doppelschleife

```
10 for i1=1 to 5
20 for i2=1 to 3
30 print ad$(i1,i2);"   ";
40 next i2
50 print
60 next i1
```

können Sie sich die Feldinhalte auf dem Bildschirm ausgeben lassen.

Die in den Programmzeilen 10 bis 50 angewandte Methode zur Eingabe der Daten ist recht umständlich und unflexibel. Etwas weiter unten werden Sie deshalb zwei wesentlich komfortablere Möglichkeiten kennenlernen.

3.3.3 Mathematische Operatoren

Wie Sie bereits wissen, beherrscht der Commodore 64 die vier Grundrechenarten Addition (+), Subtraktion (-), Multiplikation (*) und Division (/).

Hinzu kommt noch das Potenzieren (der "Hochpfeil" links neben der <Restore>-Taste; drucktechnisch bedingt wird der Hochpfeil in diesem Buch als "^" geschrieben).

```
PRINT 4^3
```

beispielsweise liefert das Ergebnis 64. Die Zahl hinter dem Pfeil gibt an, wie oft die Zahl vor dem Pfeil mit sich selbst multipliziert werden soll.

```
4^3 steht also für 4*4*4
25^7 steht für 25*25*25*25*25*25*25
```

Bei mathematischen Ausdrücken, wie beispielsweise

```
PRINT 23*4+40/8-3*5^2
```

gelten die allgemein üblichen Prioritätsregeln: Oberste Priorität hat die Potenz. Danach folgen die Multiplikation und die Division. An unterster Stelle stehen die Addition und Subtraktion.

Die obige Formel wird also wie folgt berechnet:

```
23*4+40/8-3*5^2
 92 +  5 -3*25
   97   - 75
      22
```

Das Ergebnis ist also 22. Durch gezieltes Setzen von Klammern kann man die Formel verändern. Operationen in Klammern haben Priorität vor allen anderen. Zwei Beispiele:

```
23*(4+40)/8-(3*5)^2
23*  44  /8-  15 ^2
    126.5  -  225
       -98.5

(23*4+40/(8-3*5))^2
( 92 +40/(8-15 ))^2
( 92 +40/( -7  ))^2
( 92 -  5.7     )^2
(      86.3     )^2
     7447.69
```

Für Vergleiche stehen insgesamt sechs Operatoren zur Verfügung:

```
=  gleich
<  kleiner
<= kleiner gleich
>  größer
>= größer gleich
<> ungleich
```

Der häufigste Anwendungsfall für Vergleiche ergibt sich bei IF-THEN-Befehlen, beispielsweise

```
IF A=5 THEN ....
IF W<>100 THEN ...
IF DT>=-10 THEN ...
```

Die Vergleichsoperatoren lassen sich aber auch in Formeln einsetzen.

Dazu muß man wissen, daß den beiden möglichen Vergleichsergebnissen "WAHR" oder "FALSCH" zwei Zahlen zugeordnet sind:

```
WAHR:   -1
FALSCH:  0
```

Die Formel

```
(23>10)+(10<1)
```

hätte also

```
-1+0=-1
```

zum Ergebnis. Eine besondere Bedeutung bekommt diese Tatsache in Zusammenhang mit den logischen Operatoren.

3.3.4 Logische Operatoren

Das BASIC 2.0 des Commodore 64 verfügt über drei logische Operatoren: AND, OR und NOT.

AND entspricht dem umgangssprachlichen UND, OR dem ODER und NOT einem NICHT.

Die Funktion von AND, OR und NOT macht man sich am besten an einer sogenannten Wahrheitstabelle klar.

A	B	A AND B
f	f	f
w	f	f
f	w	f
w	w	w

Das kleine "f" steht für logisch FALSCH, das kleine "w" für logisch WAHR. A AND B ergibt also nur dann WAHR, wenn auch die Bedingungen A UND B selbst WAHR sind.

A	B	A OR B
f	f	f
w	f	w
f	w	w
w	w	w

Bei A OR B genügt es, wenn eine der Bedingungen A ODER B WAHR ist.

A	NOT A
f	w
w	f

NOT kehrt der Wahrheitsgehalt einer Bedingung um. Aus WAHR wird FALSCH und umgekehrt.

Eine praktiche Bedeutung bekommen die logischen Operatoren bei IF-THEN-Abfragen.

```
IF A>1 AND A<10 THEN PRINT "A LIEGT ZWISCHEN 1 UND 10"
```

prüft beispielsweise, ob A größer als 1 UND kleiner als 10 ist.

```
IF A$="ENDE" OR A$="FERTIG" THEN END
```

beendet ein Programm, falls die Variable A$ den Text "ENDE" ODER den Text "FERTIG" enthält. Mit

```
IF NOT (A>1 AND A<10) THEN .....
```

läßt sich feststellen, ob A außerhalb des Bereichs zwischen 1 und 10 liegt. Diese Abfrage läßt sich auch anders schreiben:

```
IF A<=1 OR A>=10 THEN .....
```

Weitere Möglichkeiten, Daten zu bearbeiten, werden Sie im nächsten Abschnitt, der von den sogenannten Funktionen handeln wird, kennenlernen. Kehren wir zunächst zum Problem der Datenspeicherung zurück.

3.3.5 DATA-Zeilen

Als wir weiter vorne das Telefonnummern-Feld AD$ mit Daten auffüllten, haben wir dazu direkte Zuweisungen an die einzelnen Feldelemente verwendet. Daher mußten wir die Zuweisung "AD$(..,..)=" insgesamt 15 mal schreiben, was natürlich sehr umständlich ist. Wesentlich komfortabler wird das ganze, wenn wir sogenannte DATA-Zeilen verwenden:

```
10 data mueller,klaus,123456
20 data maier,heinz,654321
30 data schulz,georg,132456
40 data schmidt,uwe,456123
50 data mayr,peter,564321
```

Der Befehl DATA am Anfang jeder Zeile signalisiert dem Commodore 64, daß dahinter eine Liste von Daten, jeweils getrennt durch ein Komma, folgt.

Trifft der Rechner in einem Programm auf ein DATA, so überliest er den Rest der Zeile und macht mit den nächsten Anweisungen weiter. Wenn die Daten so in einer Programmzeile stehen, läßt sich nichts mit ihnen anfangen. Man muß sie erst in eine Variable oder besser in ein Variablenfeld einlesen. Dazu gibt es den Befehl READ.

```
READ TX$
```

beispielsweise weist TX$ das Datum "MUELLER" zu. READ weiß dabei durch den sogenannten DATA-Zeiger, wo die erste DATA-Zeile in einem Programm steht. READ holt aber nicht nur ein DATA-Element, sondern setzt gleichzeitig auch den DATA-Zeiger um eine Position weiter. Ein erneutes

```
READ DT$
```

weist dem Variablenfeld DT$ daher den Text "KLAUS" zu. So kann man nach und nach sämtliche in den DATA-Zeilen abgelegten Daten in Variablen einlesen. Zum Einlesen der obigen Daten in das Variablenfeld AD$ genügt eine einzige READ-Anweisung in einer Doppelschleife:

```
100 for i1=1 to 5
110 for i2=1 to 3
120 read ad$(i1,i2)
130 next i2
140 next i1
```

Beim Neustart eines Programms mit RUN wird der DATA-Zeiger immer auf das erste DATA-Element gesetzt. Häufig möchte man die Daten aber mehrfach in Variablen einlesen. In diesem Fall benutzt man den Befehl RESTORE. RESTORE setzt den DATA-Zeiger ebenfalls auf das erste Element, so daß das Einlesen wieder von vorne beginnen kann.

Wollen Sie beispielsweise die obigen Daten nacheinander in zwei Felder einlesen, so könnte das so aussehen:

```
100 for i1=1 to 5
110 for i2=1 to 3
120 read ad$(i1,i2)
130 next i2
140 next i1
150 restore:rem data-zeiger zurücksetzen
160 for i1=1 to 5
170 for i2=1 to 3
180 read tx$(i1,i2)
190 next i2
200 next i1
```

Um bei einer Erweiterung der Daten in den DATA-Feldern nicht jedesmal extra den Schleifenindex der Einleseschleife än-

dern zu müssen, gibt es übrigens einen kleinen Trick. Legen Sie doch einfach die Anzahl der Daten ebenfalls als DATA-Wert ab:

```
100 read az:rem anzahl der daten
110 for i1=1 to az
120 for i2=1 to 3
130 read ad$(i1,i2)
140 next i2
150 next i1
160 :
1000 data 5:rem anzahl
1010 data mueller,klaus,123456
1020 data maier,heinz,654321
1030 data schulz,georg,132456
1040 data schmidt,uwe,456123
1050 data mayr,peter,564321
```

Mit READ und DATA haben wir nun schon eine recht komfortable Möglichkeit, größere Datenmengen zu verwalten. Das ganze hat jedoch einen großen Nachteil:

Die Daten in den DATA-Zeilen lassen sich nur "von Hand" ändern; es gibt keine Möglichkeit, vom Programm aus Änderungen oder Ergänzungen durchzuführen.

3.3.6 Daten auf externen Speichermedien

Wer größere Datenmengen wirklich "professionell" verwalten möchte, kommt nicht umhin, sie auf einem externen Speichermedium, einer Diskette oder einer Kassette, auszulagern.

Die Arbeit mit Floppy und Datasette zählt daher mit zu den wichtigsten Programmierbereichen. Aus diesem Grund sind ihr in diesem Buch zwei Kapitel erheblichen Umfangs gewidmet. In Kapitel 5 erfahren Sie alles zum Umgang mit der Floppy, die Datasette steht im Mittelpunkt von Kapitel 10. Ich möchte Ihnen aber trotzdem gleich hier einen kleinen Einblick in die Datenspeicherung auf Diskette geben (für die Datasette gilt im Prinzip dasselbe; mehr dazu in Kapitel 10).

Größere Mengen zusammengehöriger Daten werden ganz allgemein in sogenannten "Dateien" (auch Files genannt) zusammen-

gefaßt. Auch ein auf Diskette abgespeichertes BASIC-Programm ist letztendlich nichts anderes als eine Datei, eben eine Programmdatei.

Wie bekommt man nun aber in Variablen abgelegte Daten auf die Diskette, beispielsweise unsere Telefonnummern-Daten?

Dazu muß man dem Rechner zunächst mitteilen, daß man auf eine Datei zugreifen bzw. eine neue Datei anlegen möchte. Diesen Vorgang bezeichnet man als "Öffnen" der Datei. Dazu gibt es den OPEN-Befehl:

```
10 open 1,8,2,"telefon,s,w"
```

Dadurch wird auf der Diskette eine Datei mit dem Namen "TELEFON" eingerichtet.

Die 1 ist die sogenannte logische File-Nummer, über die wir später auf die Datei Bezug nehmen werden. Die 8 signalisiert dem Rechner, daß auf die Floppy zugegriffen werden soll. Die 2 bezeichnet man als Sekundäradresse; sie soll uns hier nicht weiter interessieren.

Das "S" in dem String bedeutet, daß wir eine sequentielle Datei (und nicht etwa eine Programmdatei) öffnen wollen. Das "W" dahinter signalisiert der Floppy, daß wir in die Datei schreiben (und nicht lesen) wollen.

Das Komplizierteste haben wir damit schon hinter uns. Um die Daten nun in die Datei zu schreiben, verwenden wir eine leicht abgewandelte Form des PRINT-Befehls: PRINT#. Hinter PRINT# muß die logische File-Nummer der Datei stehen. Sie zeigt dem Rechner an, auf welche Datei wir zugreifen wollen.

```
PRINT#1,"DIESER TEXT WIRD IN DIE DATEI GESCHRIEBEN."
```

legt beispielsweise einen String in der Datei ab. Wir wollen ja aber unsere Telefondaten abspeichern. Auch das ist nicht viel aufwendiger. Wir nehmen einfach unsere bewährte Doppelschleife:

```
30 for i1=1 to 5
40 for i2=1 to 3
50 print#1,ad$(i1,i2)
60 next i2
70 next i1
```

Was man öffnet, muß man irgendwann auch wieder schließen. Schließen wir also auch die Datei "TELEFON". Dazu gibt es den Befehl CLOSE:

```
90 close 1
```

Die 1 ist wieder die logische File-Nummer. Schön und gut, die Daten sind nun auf Diskette abgelegt. Doch, wie kommt man wieder an sie heran? Dazu muß die Datei wieder geöffnet werden, diesmal aber nicht zum Schreiben, sondern zum Lesen:

```
110 open 1,8,2,"telefon,s,r"
```

Anschließend lesen wir die Daten mit einer abgewandelten Form des INPUT-Befehls: INPUT#. Hinter INPUT# muß wieder die logische File-Nummer der Datei stehen:

```
130 for i1=1 to 5
140 for i2=1 to 3
150 input#1,ad$(i1,i2)
160 next i2
170 next i1
```

Zum Schluß vergessen wir nicht, die Datei zu schließen:

```
190 close 1
```

Fassen wir das ganze noch einmal zusammen:

```
5 rem sequentielle datei schreiben
10 open 1,8,2,"telefon,s,w"
20 :
30 for i1=1 to 5
40 for i2=1 to 3
50 print#1,ad$(i1,i2)
60 next i2
70 next i1
80 :
90 close 1
95 :
```

```
97 :
100 rem sequentielle datei lesen
110 open 1,8,2,"telefon,s,r"
120 :
130 for i1=1 to 5
140 for i2=1 to 3
150 input#1,ad$(i1,i2)
160 next i2
170 next i1
180 :
190 close 1
```

Sie sehen, das ganze folgt einem doch recht einleuchtenden Prinzip:

```
Datei öffnen
Daten schreiben oder lesen
Datei schließen
```

Zum Schluß möchte ich noch kurz auf zwei spezielle Eingabebefehle eingehen: GET# und GET. Wenn Sie bei INPUT auf dem Bildschirm Daten eingeben, werden diese ja erst übernommen, sobald Sie die <Return>-Taste drücken. Wenn nun aber INPUT# aus einer Datei Daten liest, woher weiß der Befehl eigentlich, wo das eine Datum aufhört und das nächste beginnt? Die Daten sind in der Datei ja der Reihe nach abgelegt:

```
MUELLER KLAUS 123456 MAIER .....
```

Nun, auch der PRINT#-Befehl, mit dem wir die Daten in die Datei geschrieben haben, "drückt" am Ende jedes Datums die <Return>-Taste, indem er einen speziellen Code, den ASCII-Code 13, sendet, der dann in der Datei hinter dem Datum abgelegt wird. Sobald INPUT# auf diesen Code trifft, weiß es, daß es das aktuelle Datum vollständig gelesen hat.

In manchen Fällen möchte man die Daten aber gar nicht so "häppchenweise", sondern lieber Zeichen für Zeichen einlesen. Hier kommt der Befehl GET# ins Spiel:

```
10 open 1,8,2,"telefon,s,r"
20 get eg$
30 close 1
```

liest nur den ersten Buchstaben "M" von MUELLER in die Variable EG$. Mit der Schleife

```
10 open 1,8,2,"telefon,s,r"
20 for z=1 to 7:get eg$:dt$=dt$+eg$:next z
30 close 1
```

wird der gesamte Name MUELLER gelesen und in der Variablen DT$ abgelegt. Mit GET# kann man also nach und nach die gesamte Datei einlesen und in beliebige "Happen" unterteilen.

Auch für das Bildschirm-GET gibt es sinnvolle Anwendungsfälle. Häufig hat man ja in einem Programm nur Einzelzeichen-Abfragen, etwa die Kennziffer bei einem Auswahlmenü, oder eine Ja-Nein-Abfrage, wie z.B:

```
10 print "programm beenden? (j/n)"
20 input e$
30 if e$="j" then end
40 rem weiterer programmverlauf
Mit GET läßt sich das so umschreiben:
10 print "programm beenden? (j/n)"
20 get e$:if e$="" then 20
30 if e$="j" then end
40 rem weiterer programmverlauf
```

GET bringt dem Anwender des Programms den Vorteil, daß er nicht extra, wie bei INPUT, die <Return>-Taste drücken muß.

Die Programmzeile 20 zeigt gleichzeitig eine andere sehr häufige Anwendung von GET: Zeile 20 wartet auf einen (beliebigen) Tastendruck!

3.4 Funktionen

Den Begriff "Funktion" werden Sie wahrscheinlich noch aus dem Mathematikunterricht kennen. Eine Funktion berechnet aus einem vorgegebenen Wert, dem sogenannten Argument der Funktion, einen neuen Wert, den Funktionswert. Nehmen wir ein ganz einfaches Beispiel:

```
F(X)=X*X
```

Diese Funktion mit dem Namen "F" berechnet die zweite Potenz des Wertes X. Für die X-Werte 1 bis 5 ergeben sich folgende Funktionswerte F:

X	F
1	1
2	4
3	9
4	16
5	25

Analog dazu arbeiten auch alle anderen Funktionen. Es gibt im BASIC 2.0 aber eine besondere Gruppe von Funktionen, die etwas aus dem Rahmen fallen: die sogenannten String-Funktionen. Im Gegensatz zu den numerischen Funktionen, bei denen nur Zahlenwerte als Argument erlaubt sind, kann man mit Hilfe der String-Funktionen alphanumerische Ausdrücke bearbeiten.

3.4.1 Numerische Funktionen

Das BASIC 2.0 verfügt über eine Vielzahl vordefinierter Funktionen, hauptsächlich aus dem mathematischen Bereich.

Die Funktionen INT, SGN und ABS

INT berechnet den ganzzahligen Anteil des numerischen Wertes.

```
PRINT INT(126,78)   ergibt 126
PRINT INT(85.126)   ergibt 85
PRINT INT(-371.412) ergibt -372
PRINT INT(-53.91)   ergibt -54
```

INT eignet sich auch sehr gut zur Rundung von Zahlen. Dazu addiert man zu der Zahl jeweils 0.5:

```
10 rem rundung
20 input zl
30 zl=int(zl+0.5)
40 print zl
```

Bei einem Nachkommawert unter 0.5 wird ZL auf die nächstniedrigere Stelle abgerundet. Bei einem Nachkommawert größer gleich 0.5 dagegen wird ZL auf die nächsthöhere Stelle aufgerundet.

SGN ermittelt das Vorzeichen eines numerischen Wertes. SGN kann nur die drei Werte -1, 0 und 1 annehmen, entsprechend dem Vorzeichen des Arguments:

```
X<0: SGN(X)=-1
X=0: SGN(X)=0
X>0: SGN(X)=1
```

Die Funktion ABS ermittelt den Betrag einer Zahl.

```
PRINT ABS(6352)  ergibt 6352
PRINT ABS(-423)  ergibt 423
PRINT ABS(-5.98) ergibt 5.98
```

ABS macht also aus einer negativen eine positive Zahl. Das ist zum Beispiel dann sehr nützlich, wenn man bei einem komplizierten mathematischen Ausdruck nicht ganz sicher ist, ob dieser immer positiv ist, dies aber für die Weiterverarbeitung des Ausdrucks zwingend notwendig ist. In diesem Fall setzt man den gesamten Ausdruck einfach in ein ABS(.....).

Zufallszahlen berechnen

Nicht nur in Spielen, auch in anderen Programmen benötigt man ab und zu Zahlen, die "zufällig" ermittelt worden sind. Natürlich kann eine Zahl, die von einem Computer berechnet wird, niemals wirklich "zufällig" sein. Es gibt aber spezielle Berechnungsverfahren, die eine annähernd zufällige Verteilung der Zahlenwerte erlauben. Das BASIC 2.0 verfügt dazu über die Funktion RND. RND liefert Zufallszahlen zwischen (ausschließlich) 0 und 1.

In den meisten Fällen möchte man aber Zufallszahlen in einem anderen Bereich, etwa zwischen 1 und 10 oder zwischen 100 und 150, erzeugen lassen. Dazu gibt es eine einfache Umrechnungsformel:

```
10 rem zufallszahlen erzeugen
20 ug=1:rem untere grenze
30 og=10:rem obere grenze
40 z1=ug+rnd(0)*(og-ug)
50 z2=int(z1)
```

Z1 enthält jeweils eine Zufallszahl mit Nachkommateil. In Zeile 50 wird der Nachkommateil abgeschnitten. Z2 enthält also immer ganzzahlige Zufallszahlen, die man in der Regel auch am häufigsten benötigt.

In den Zeilen 20 und 30 können Sie den Bereich, in dem die Zufallszahlen erzeugt werden sollen, beliebig festlegen.

Eine besondere Bewandtnis hat es mit dem Argument von RND. Der Wert des Arguments bestimmt nämlich, woher der Startwert für die Berechnungsformel zur Erzeugung der Zufallszahlen genommen wird.

Das Argument ist positiv

RND beginnt (nach dem Einschalten des Commodore 64) immer mit demselben Startwert, der an einer bestimmten Stelle des Rechnerspeichers fest abgelegt ist.

RND(1) erzeugt also immer wieder dieselben Zufallszahlen, vorausgesetzt der Rechner wird zwischen zwei Programmläufen ausgeschaltet.

Das Argument ist negativ

In diesem Fall berechnet RND den Startwert unter Einbeziehung des Argumentwerts. RND(-1) beispielsweise erzeugt also immer denselben Zufallswert, während beispielsweise RND(-1000) zwar einen völlig anderen Wert berechnet, dafür aber auch immer wieder denselben.

RND mehrmals hintereinander mit demselben negativen Argument zu benutzen, ergibt daher keinen Sinn. RND(-...) eignet sich aber sehr gut dazu, die Zufallszahlenfolge mit einer bestimmten Zufallszahl zu initialisieren.

Das Argument ist Null

Bei RND(0) holt sich der Rechner den Startwert aus einem internen Baustein zur Steuerung des Commodore 64, der ständig wechselnde Werte enthält.

Mit RND(0) erhält man daher noch am ehesten zufällige Werte.

Den freien Speicherplatz ermitteln

Für BASIC-Programme stehen Ihnen insgesamt 38911 Speicherplätze zur Verfügung. Die Anzahl der Speicherplätze bezeichnet man in der Regel mit "Bytes", wie Sie auch an der Einschaltmeldung des Commodore 64 (38911 BASIC BYTES FREE) ersehen können.

Möchte man nun erfahren, wieviel Speicherplatz einem im Augenblick noch zur Verfügung steht, dann nimmt man die Funktion FRE.

Das Funktionsargument hat bei FRE keinerlei Bedeutung, muß aber angegeben werden. Ob Sie also schreiben FRE(123) oder FRE(-321), ist völlig egal. Am einfachsten ist es, man setzt eine Null ein.

```
PRINT FRE(0)
```

gibt also die Größe des freien BASIC-Speichers aus. FRE hat allerdings einen Fehler, der in manchen Fällen zu einem negativen Wert führt, zu dem man dann 65535 addieren muß, um den korrekten Wert zu erhalten.

Den Cursor positionieren

Wer schon mit einer Schreibmaschine gearbeitet hat, wird den sogenannten "Tabulator" kennen, mit dem man bestimmte Spaltenpositionen markieren kann, die dann mit einem Tastendruck angesteuert werden können. Sehr nützlich ist das beispielsweise beim Schreiben von Tabellen. Beim Commodore 64 hat man gleich zwei Tabulatorfunktionen: TAB und POS.

TAB setzt den Cursor immer auf die in den dahinterstehenden Klammern angegebene Spaltenposition.

```
PRINT TAB(10) "HALLO"
```

druckt beispielsweise den Text "HALLO" ab Spalte 10. Im Unterschied dazu setzt SPC den Cursor um die dahinter angegebenen Spaltenpositionen weiter.

```
PRINT "HALLO1" SPC(10) "HALLO2"
```

setzt das "HALLO2" also genau 10 Spalten hinter das "HALLO1". Der Vorteil gegenüber einem

```
PRINT "HALLO1          HALLO2"
```

ist natürlich der, daß bei SPC die evtl. zwischen den beiden Texten bereits auf dem Bildschirm stehenden Zeichen nicht überschrieben werden.

Die Funktion SQR

Mit SQR kommen wir zu den rein mathematischen Funktionen. SQR berechnet die Quadratwurzel eines Wertes, also diejenige Zahl, die mit sich selbst multipliziert den Ausgangswert ergibt.

```
PRINT SQR(4)
```

ergibt beispielsweise 2. Das Argument von SQR muß positiv sein. Ein typischer Anwendungsfall ist die sogenannte "Mitternachtsformel" zur Lösung der folgenden algebraischen Gleichung AX^2+BX+C=0:

```
10 a=...
20 b=...
30 c=...
40 :
50 x1=(-b+sqr(b*b-4*a*c)/(2*a)
60 x2=(-b-sqr(b*b-4*a*c)/(2*a)
```

X1 und X2 enthalten die beiden Lösungen der Gleichung.

Die Funktionen EXP und LOG

EXP ermittelt den sogenannten "natürlichen Exponenten" des angegebenen Arguments. EXP(X) könnte man auch als

```
2.718^X
```

schreiben, wobei 2.718 eine Annäherung an die sogenannte "Eulersche Zahl" ist, die in der Mathematik eine große Rolle spielt.

Der Wertebereich für das Argument X von EXP ist auf etwa -88 bis +88 beschränkt, da EXP mit zunehmenden Argumentwerten sehr stark anwächst. LOG berechnet den sogenannten "natürlichen Logarithmus" des Argumentwertes, stellt also die Umkehrfunktion zu EXP dar.

```
LOG(10)
```

ist also diejenige Zahl, mit der man 2.718 potenzieren muß, um die Zahl 10 zu erhalten. Das Argument von LOG(X) muß positiv sein.

In manchen Fällen benötigt man den Logarithmus zu einer anderen Basis, etwa der Basis 10. Hierzu gibt es eine einfache Berechnungsformel:

```
LG=LOG(X)/LOG(10)
```

Indem man in das zweite LOG also die gewünschte Basis einsetzt, kann man für X den Logarithmus zu jeder beliebigen Basis berechnen.

Die trigonometrischen Funktionen SIN, COS, TAN und ATN

SIN(X) berechnet den Sinus des Arguments X, COS(X) den Kosinus von X, TAN(X) den Tangens von X.

ATN(X) ist die Umkehrfunktion zu SIN(X), sie berechnet den sogenannten Arcussinus von X. Die Argumente müssen bei allen Funktionen im sogenannten "Bogenmaß" angegeben werden. Eine

andere bei den trigonometrischen Funktionen gebräuchliche Einheit ist "Grad", die Einteilung geht dabei von 0 Grad bis 360 Grad.

Zur Umrechnung von Grad in Bogenmaß und umgekehrt gibt es zwei einfache Formeln:

```
Grad -> Bogenmaß: BG=GR*3.14/180
Bogenmaß -> Grad: GR=BG*180/3.14
```

Die 3.14 steht annäherungsweise für die mathematische Konstante PI.

```
S=SIN(45*3.14/180)
```

berechnet also den Sinus von 45 Grad.

```
C=COS(90*3.14/180)
```

ermittelt den Cosinus von 90 Grad.

Da die Sinus- und die Cosinus-Funktion nur Werte zwischen -1 und 1 annehmen, kann man sich die beiden Funktionen leicht grafisch darstellen lassen.

Das folgende Programm druckt eine senkrechte Sinuskurve auf den Bildschirm:

```
10 for x=0 to 6 step 0.29
20 p=int(20+18*sin(x))
30 print tab(p) "*"
40 next x
```

Wenn Sie die Zeile 20 gegen

```
20 p=int(20+18*cos(x))
```

austauschen, erhalten Sie eine Kosinuskurve, die gegenüber der Sinuskurve um 90 Grad verschoben ist.

3.4.2 String-Funktionen

Wie Sie bereits wissen, ist es möglich, zwei Strings miteinander zu "addieren". Dabei werden die beiden Strings einfach aneinander gesetzt, und es entsteht ein neuer String.

```
"COMMO"+"DORE "+" 64"
```

beispielsweise ergibt

```
"COMMODORE 64"
```

Die Funktionen CHR$ und ASC

Wie ich bereits mehrfach erwähnt habe, legt der Rechner die Strings im sogenannten ASCII-Code im Speicher ab. Dieses Codes kann man sich auch selbst bedienen. Dazu gibt es die Funktion CHR$.

CHR$ liefert das zu dem angegebenen Code zugehörige Zeichen.

```
CHR$(65)
```

beispielsweise ein "A". Anstelle von

```
PRINT "A"
```

könnte man deshalb auch schreiben

```
PRINT CHR$(65)
```

Interessant wird CHR$ im Zusammenhang mit den sogenannten Steuercodes. Der ASCII-Code 147 beispielsweise löscht den Bildschirm, der Code 157 bewegt den Bildschirm-Cursor um eine Stelle nach links. Die folgende Tabelle enthält eine kleine Zusammenstellung wichtiger Steuercodes, eine vollständige ASCII-Tabelle finden sie im Anhang:

```
Revers ein: 18
Revers aus: 146
Cursor ab: 17
Cursor hoch: 145
Cursor rechts: 29
```

```
Cursor links: 157
Cursor Home: 19
Bildschirm löschen: 147
Auf Groß-/Kleinschrift umschalten: 14
Auf Grafik-/Großschrift umschalten: 142
```

Wollen Sie beispielsweise einen Text in Negativschrift ausgeben lassen, so schreiben Sie einfach:

```
PRINT CHR$(18);"NEGATIVSCHRIFT";CHR$(146)
```

Sehr nützlich sind auch die Cursor-Steuercodes. Häufig möchte man bei einer Dateneingabe mit INPUT eine Standardeingabe vorgeben, so daß der Anwender nur noch die <Return>-Taste drücken muß. Das könnte zum Beispiel so aussehen:

```
10 print "wieviele daten:   100";
.....
20 for z=1 to 6:print chr$(157);:next z
.....
30 input eg
.....
```

Als Vorgabe für die Eingabe erhält der Anwender die Zahl 100. Damit nun aber der Cursor bei INPUT an der richtigen Stelle, d.h. direkt hinter dem Doppelpunkt steht, wird der Cursor in Zeile 20 um sechs Stellen nach links bewegt.

Das Gegenstück zu CHR$ bildet die Funktion ASC. ASC ermittelt zu einem vorgegebenen Zeichen dessen ASCII-Code.

```
PRINT ASC("A")
```

beispielsweise liefert den Wert 65. Auch ASC hat einen unmittelbaren praktischen Nutzen. Vielleicht haben Sie sich in der Zwischenzeit auch schon gefragt, wie man die vier sogenannten Funktionstasten rechts auf der Tastatur abfragen könnte.

Die Funktionstasten eignen sich ja beispielsweise sehr gut zur Funktionsauswahl aus einem Auswahlmenü, wie man es auch häufig bei kommerziellen Programmen sieht. Ein Programm-Menü könnte zum Beispiel so aufgebaut sein:

```
F1: DATEN EINGEBEN
F2: DATEN AUSGEBEN
F3: DATEN SORTIEREN
F4: DATEN DRUCKEN
F5: DATEN SUCHEN
F6: FLOPPY ANSTEUERN
F7: DATASETTE ANSTEUERN
F8: PROGRAMM BEENDEN
```

Jeder Funktionstaste ist nun ein bestimmter ASCII-Code zugeordnet:

```
F1: 133
F2: 137
F3: 134
F4: 138
F5: 135
F6: 139
F7: 136
F8: 140
```

Die Funktionstasten F2, F4, F6 und F8 erreichen Sie übrigens durch gleichzeitiges Drücken der betreffenden Taste und einer der beiden <Shift>-Tasten. Mit dem Wissen um diese Codes und der Funktion ASC läßt sich die Abfrage nun leicht realisieren:

```
100 print "auswahlmenue:"
110 print "f1: daten eingeben"
120 print "f2: daten ausgeben"
130 print "f3: daten sortieren"
140 print "f4: daten drucken"
150 print "f5: daten suchen"
160 print "f6: floppy ansteuern"
170 print "f7: datasette ansteuern"
180 print "f8: programm beenden"
190:
195 rem tastaturabfrage
200 get eg$:if eg$="" then 200
210 eg=asc(eg$)
220 :
230 if eg=133 then gosub ....:rem funktionstaste 1
240 if eg=137 then gosub ....:rem funktionstaste 2
250 if eg=134 then gosub ....:rem funktionstaste 3
260 if eg=138 then gosub ....:rem funktionstaste 4
270 if eg=135 then gosub ....:rem funktionstaste 5
280 if eg=139 then gosub ....:rem funktionstaste 6
290 if eg=136 then gosub ....:rem funktionstaste 7
300 if eg=140 then gosub ....:rem funktionstaste 8
310 :
320 goto 200
```

Die Funktion LEN

Sehr häufig möchte man wissen, wie viele Zeichen ein String enthält. Dazu gibt es die Funktion LEN:

```
PRINT LEN("12345")
```

ergibt beispielsweise 5. LEN ist es dabei völlig egal, welche Art von Zeichen der String enthält (Buchstaben, Ziffern, Leerzeichen, Steuerzeichen). Es zählt einfach die Anzahl der Speicherplätze, die der String im Rechnerspeicher belegt.

```
PRINT LEN("B E I S P I E L") ergibt 15

PRINT LEN("") ergibt 0
```

Natürlich darf man auch String-Variablen verwenden:

```
A$="TEXT":PRINT LEN(A$) ergibt 4
```

Die Funktionen STR$ und VAL

Oft steht man vor dem Problem, daß numerische Daten formatiert ausgegeben werden sollen, beispielsweise rechtsbündig:

```
12846
   18
  394
    7
 3646
```

Wesentlich dabei ist, daß man die Länge der betreffenden Zahlen, also ihre Stellenzahl kennt. Nun könnte man die Länge natürlich über die (numerische) Größe der Zahl ermitteln, etwa mit:

```
IF Z<10 then .....:rem einstellig
IF Z<100 THEN .....:rem zweistellig
IF Z<1000 THEN .....:rem dreistellig
```

Wesentlich einfacher wird es aber, wenn man die Zahl in einen String umwandelt. Dazu gibt es die Funktion STR$:

```
T$=STR$(12345)
```

beispielsweise weist T$ den String

```
" 12345"
```

zu (Leerstelle vor der 1 wird wegen des Vorzeichens eingefügt).

```
T$=STR$(-123.45)
```

ergäbe

```
"-123.45"
```

Mit Hilfe von STR$ und LEN lassen sich Zahlen nun leicht formatieren, etwa rechtsbündig ausgeben:

```
10 z(1)=12846
20 z(2)=18
30 z(3)=394
40 z(4)=7
50 z(5)=3646
60 :
100 for i=1 to 5
110 ft$=str$(z(i)):rem zahl in string umwandeln
120 if len(ft$)<10 then ft$=" "+ft$:goto 120
130 print ft$
140 next i
```

Der eigentliche Formatiervorgang findet in Programmzeile 120 statt. Der String FT$ wird so lange mit Leerstellen aufgefüllt, bis er die Länge 10 hat. Das Gegenstück zu STR$ bildet die Funktion VAL. VAL wandelt einen String in einen numerischen Wert um, sofern der String aus Ziffern besteht:

```
A=VAL("3591")
```

beispielsweise weist A den Wert 3591 zu.

```
B=VAL("-587.32")
```

weist B den Wert -587.32 zu.

VAL ist erfreulicherweise so "tolerant", daß es Nichtziffernzeichen am Ende des Strings akzeptiert.

```
C=VAL("10 TEILE")
```

weist C daher den Wert 10 zu.

Enthält der betreffende String überhaupt keine Ziffern, dann erhält die Variable den Wert Null. Beispielsweise ergibt

```
T$="TEXT":WT=VAL(T$)
WT=0.
```

Die Funktionen LEFT$, MID$ und RIGHT$

Die mit Sicherheit interessantesten String-Funktionen sind LEFT$, MID$ und RIGHT$. Damit kann man aus einem vorhandenen String Teil-Strings "herausschneiden".

LEFT$ schneidet dabei vom linken Rand eines Strings Zeichen heraus.

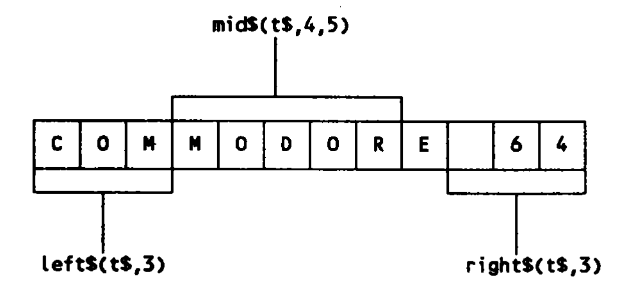
RIGHT$ schneidet auf der rechten Seite Teil-Strings ab.

MID$ schließlich holt sich einen Teil-String aus der Mitte heraus.

Zerschneiden wir als Beispiel einmal den String

```
T$="COMMODORE 64":
PRINT LEFT$(T$,3)  ergibt "COM"
PRINT MID$(T$,4,5) ergibt "MODOR"
PRINT RIGHT$(T$,3) ergibt " 64"
```

Zur Verdeutlichung das ganze einmal grafisch dargestellt:



Natürlich darf man für die Positions- und Längenangaben auch Variablen einsetzen:

```
10 t$="commodore"
20 :
30 for p=1 to len (t$)
40 print left$(t$,p)
50 next p
```

Dieses kleine Programm erzeugt die folgende Ausgabe:

```
C
CO
COM
COMM
COMMO
COMMOD
COMMODO
COMMODOR
COMMODORE
```

3.4.3 Funktionen selbst definieren

Wem die vorhandenen Funktionen nicht ausreichen, der kann sich auch - in gewissem Umfang - Funktionen selbst definieren, allerdings nur numerische. Dazu hält BASIC einen speziellen Befehl parat: DEF. Hinter DEF folgt der gewünschte Name der Funktion, der immer mit einem FN beginnt und ansonsten denselben Regeln unterliegt wie die Variablennamen. Hinter dem Namen steht in Klammern die Argumentvariable der Funktion. Dahinter schließlich, getrennt durch ein Gleichheitszeichen, die eigentliche Funktionsdefinition.

Sehen wir uns das ganze einmal an einem praktischen Beispiel an. Die Fläche eines Kreises berechnet sich bekanntlich nach der Formel:

```
KF=R*R*3.14
```

wobei R für den Radius des Kreises steht. Definieren wir also eine Funktion KF, die eben diese Berechnung durchführt:

```
DEF FN KF(R)=R*R*3.14
```

Die Variable R ist in diesem Fall das Argument der Funktion. Aufrufen läßt sich die Funktion KF nun wie jede andere vordefinierte Funktion:

```
PRINT FN KF(5)
```

ermittelt die Fläche eines Kreises mit dem Radius 5. Sehr wichtig ist, daß die Funktionsdefinition mit DEF vor dem ersten Funktionsaufruf abgearbeitet wird. Die Definition schreibt man daher am besten ganz an den Anfang eines Programms oder in ein Unterprogramm, das dann am Programmanfang aufgerufen wird.

Mit Hilfe eines Unterprogramms können Sie sich leicht eine ganze Sammlung von Funktionen anlegen, die Sie dann bei Bedarf an Ihre Programme anhängen.

Was man beispielsweise bei mathematischen und auch bei grafischen Anwendungen immer wieder benötigt, sind spezielle trigonometrische und hyperbolische Funktionen, die im BASIC 2.0 nicht vorhanden sind. Diese Funktionen lassen sich aber aus den implementierten Funktionen SIN, COS usw. herleiten beispielsweise:

```
Arcussinus:          ATN(X/SQR(1-X^2)
Arcuscosinus:        -ATN((X/SQR(1-X^2))+3.14/2
Sinus Hyperbolicus: (EXP(X)-EXP(-X))/2
```

Definieren wir uns diese Funktionen selbst:

```
1000 rem selbstdefinierte funktionen
1010 :
1015 rem arcussinus
1020 def fn as(x)=atn(x/sqr(1-x^2)
1025 rem arcuscosinus
1030 def fn ac(x)=-atn((x/sqr(1-x^2))+3.14/2
1035 rem sinus hyperbolicus
1040 def fn sh(x)=(exp(x)-exp(-x))/2
1490 :
1500 return
```

Wenn Sie dieses Unterprogramm am Programmanfang mit GOSUB 1000 aufrufen, werden automatisch alle Funktionen definiert.

3.5 Fortgeschrittene Techniken

Insbesondere bei größeren Programmen schleichen sich oft Programmfehler ein, die nur sehr schwer zu finden sind. Da wäre man manchmal froh, es würde sich nur um einen einfachen Tippfehler handeln, über den man sich normalerweise ja so leicht ärgert.

Häufig liegt die Fehlerursache in einer falsch geschriebenen Formel oder in einer falschen Variablenzuweisung. Arbeitet man beispielsweise mit vier Variablen mit den Namen P1, P2, P3 und P4, denen im Programm sehr oft Werte zugewiesen werden, so kann es leicht passieren, daß man vielleicht anstelle von P3=7 (wie es von der Programmfunktion her richtig wäre) in der Eile schreibt P2=7. Auch wenn man das Programm dann später noch so sorgfältig durchliest, entdeckt man solche Flüchtigkeitsfehler erfahrungsgemäß nur sehr schwer.

Kurz und gut, man müßte eine Möglichkeit haben, sich die Inhalte einzelner Variablen jederzeit während des Programmablaufs anzuschauen. Diese Möglichkeit gibt es! Zunächst einmal muß man den Programmablauf unterbrechen. Dazu dient der Befehl STOP. Sobald der Rechner in einem Programm auf ein STOP trifft, bricht er die Programmabarbeitung ab und kehrt in den Direktmodus zurück. Im Direktmodus können Sie sich nun mittels PRINT den Inhalt jeder beliebigen Variablen ausgeben lassen. Anschließend - und das ist das wesentliche - können Sie die Programmabarbeitung durch den Befehl CONT fortsetzen lassen!

Geben Sie also - im Direktmodus - CONT ein, dann fährt der Rechner hinter der STOP-Anweisung im Programm fort. Diesen Vorgang dürfen Sie, sooft Sie wollen, wiederholen. Die einzige Einschränkung: Während das Programm unterbrochen ist und der Rechner sich im Direktmodus befindet, dürfen Sie an dem Pro-

gramm selbst keine Veränderungen vornehmen. Sobald Sie eine Programmzeile ändern und danach CONT eingeben, reagiert der Rechner mit einem CAN'T CONTINUE ERROR. In diesem Fall müssen Sie das Programm mit RUN neu starten. Sehen wir uns das ganze an einem praktischen Beispiel an, unserem Rechnungsprogramm aus dem ersten Abschnitt:

```
90 rem programm: rechnung
92 :
94 :
98 :
100 print "rechnung erstellen"
110 print "——————————————"
112 :
113 :
115 rem preise eingeben
117 :
120 input "preis 1: ";p1
130 input "preis 2: ";p2
140 input "preis 3: ";p3
142 :
143 :
145 rem mehrwertsteuer berechnen
147 :
150 m1=p1*0.14
160 m2=p2*0.14
170 m3=p3*0.14
180 mw=m1+m2+m3:stop:rem unterbrechnung 1
182 :
183 :
185 rem gesamtpreis berechnen
187 :
190 gp=p1+p2+p3
200 ig=gp+mw:stop:rem unterbrechung 2
202 :
203 :
205 rem ergebnisse ausgeben
207 :
210 print "——————————————"
220 print "preis 1: ";p1;"  mehrwertsteuer: ";m1
230 print "preis 2: ";p2;"  mehrwertsteuer: ";m2
240 print "preis 3: ";p3;"  mehrwertsteuer: ";m3
250 print "——————————————"
260 print "gesamtpreis: ";gp
270 print "mehrwertsteuer:";mw
280 print "insgesamt:";ig
282 :
284 :
286 :
290 end
```

Die erste Unterbrechung findet in Zeile 180 statt. Anschließend lassen Sie sich beispielsweise die Inhalte der Variablen M1, M2, M3 und MW ausgeben. Nachdem Sie CONT eingegeben haben, läuft das Programm bis zur Zeile 200, wo die zweite Unterbrechung stattfindet. Nun lassen Sie sich vielleicht die Inhalte von GP und IG ausgeben. Nach einem zweiten CONT läuft das Programm dann bis zum Ende durch.

Im Extremfall könnten Sie hinter jeden einzelnen Programmbefehl ein STOP einfügen. Bitte vergessen Sie aber eines nicht: Sobald Ihr Programm korrekt läuft, müssen Sie natürlich alle STOP-Befehle wieder entfernen!

3.6 Den Drucker ansteuern

Oft möchte man die Ausgaben eines Programms nicht nur auf dem Bildschirm haben, sondern auch Schwarz auf Weiß auf Papier. Um den Drucker von einem Programm aus anzusteuern, geht man ähnlich vor wie beim Zugriff auf die Floppy. Wegen der vielen verschiedenen Druckerfabrikate gibt es allerdings zum Teil erhebliche Unterschiede im Detail, etwa wenn es darum geht, eine bestimmte Schriftart einzustellen oder auf Grafikdruck umzuschalten.

Ich möchte mich deshalb an dieser Stelle auf allgemeine Dinge beschränken, die für jeden Drucker (ausgenommen vielleicht ganz exotische Geräte) gelten. Zunächst zur Geräteadresse. Der Drucker hat die Geräteadresse 4. Um den Drucker anzusprechen, muß man daher eine logische Datei wie folgt öffnen:

```
OPEN 4,4
```

Für die logische File-Nummer dürfen Sie natürlich auch einen anderen Wert wählen. Am einfachsten ist es aber, man nimmt ebenfalls eine 4. Bei der sogenannten Sekundäradresse, die ja wahlweise hinter der Geräteadresse angegeben werden kann, fangen die Probleme schon an. Die Commodore-Drucker (MPS 801 u.a.) schalten bei der Sekundäradresse 0, also durch

```
OPEN 4,4,0
```

auf Grafik-/Großschrift um, bei der Sekundäradresse 7 dagegen, also durch

```
OPEN 4,4,7
```

auf Groß-/Kleinschrift. Diese beiden Zeichensätze entsprechen den Zeichensätzen am Bildschirm, zwischen denen Sie ja durch gleichzeitiges Drücken der <Commodore>- und der <Shift>-Taste umschalten können.

Nebenbei bemerkt: Auch wenn die Schriftqualität bei den Commodore-Druckern nicht die allerbeste ist, einen großen Vorteil haben Sie. Nur mit einem Commodore-Drucker läßt sich der gesamte Grafikzeichensatz zu Papier bringen. Bei den anderen Druckerfabrikaten sind die Grafikzeichen in der Regel nicht vorhanden oder können allenfalls nur mit hohem Aufwand gedruckt werden.

Bei Druckern anderer Hersteller ruft eine bestimmte Sekundäradresse zum Teil sehr unterschiedliche Reaktionen hervor. Im Zweifelsfall läßt man sie einfach weg oder schaut im Handbuch zum Drucker nach. Nachdem die Datei geöffnet ist, kann man mit dem schon bekannten

```
PRINT#4,.....
```

die auszudruckenden Daten an den Drucker schicken. Zum Schluß muß die Datei wieder, wie gewohnt, mit

```
CLOSE 4
```

geschlossen werden. Eine typische Druckerausgabe könnte also beispielsweise so aussehen:

```
10 open 4,4
20 print#4,"zu druckender text";dr;dr$
30 close 4
```

Interessanter wird es, wenn man eine bestimmte Sonderfunktion des Druckers einstellen möchte. Wie teilt man dies dem Drucker mit?

Dazu muß man die Ihnen bereits bekannte Funktion CHR$ bemühen. Mit CHR$ wird der die gewünschte Sonderfunktion aktivierende Code an den Drucker gesendet. Bei Commodore-Druckern kann man beispielsweise mit

```
CHR$(18)
```

auf Negativschrift umschalten.

```
CHR$(146)
```

schaltet wieder auf Normalschrift um. Ein Beispiel:

```
10 open 4,4
20 print#4,chr$(18);"invers";chr$(146);"normal"
30 close 4
```

Bei anderen Druckerfabrikaten, die in der Regel nach dem sogenannten "ESC/P-Standard" arbeiten, muß man dem Drucker die Steuercodes in einer sogenannten ESC-Sequenz (Escape-Sequenz) mitteilen. Eine ESC-Sequenz wird immer mit

```
CHR$(27)
```

eingeleitet. Alles, was danach kommt, wird im Drucker von einer speziellen Routine ausgewertet und dann in die entsprechende Druckerfunktion umgesetzt. Ein Beispiel:

```
10 open 4,4
20 print#4,chr$(27);chr$(69);"fettschrift"
30 close 4
```

Durch "CHR$(27);CHR$(69)" wird also die Fettschrift aktiviert (natürlich nur bei einem ESC/P-Drucker).

Eine weitere Möglichkeit, Daten ausdrucken zu lassen, haben Sie bereits im ersten Abschnitt dieses Kapitels beim Ausdrucken eines Programm-Listings kennengelernt: den Befehl CMD. CMD

dient dazu, die Bildschirmausgabe auf ein anderes Gerät (das könnte auch die Floppy sein) "umzulenken". Wenn Sie in einem Programm also schreiben:

```
10 open 4,4
20 cmd 4
.....
100 print#4
110 close 4
```

Dann wirken sämtliche zwischen den Programmzeilen 20 und 100 stehenden PRINT-Anweisungen auf den Drucker. Dieses Verfahren ist eine bequeme Methode, zwischen der Bildschirm- und der Druckerausgabe wahlweise umzuschalten. Wenn Sie das ganze nämlich durch eine Abfrage ergänzen, können Sie mit denselben PRINT-Anweisungen sowohl auf den Bildschirm als auch auf den Drucker zugreifen:

```
10 input "daten auf drucker ausgeben? (j/n) ",an$
20 if an$="n" then 40
30 open 4,4:cmd 4
40 rem weiteres programm
.....
100 if an$="j" then print#4:close 4
```

Zu diesem Problem gibt es aber noch eine elegantere Lösung. Dazu muß man wissen, daß auch dem Bildschirm eine Geräteadresse zugeordnet ist: die Adresse 3! Deshalb gibt man die Daten einfach grundsätzlich über eine logische Datei aus:

```
10 input "daten auf drucker ausgeben? (j/n) ",an$
20 if an$="n" then 40
30 open 4,4:goto 50:rem drucker-file öffnen
40 open 4,3:rem bildschirm-file öffnen
50 rem weiteres programm
.....
70 print#4,.....
.....
100 close 4
```

Durch eine entsprechende Änderung der OPEN-Anweisung können Sie die Bildschirmausgabe jetzt auf jedes beliebige Gerät (mit OPEN 4,8 beispielsweise auf die Floppy) umlenken, ohne auch nur eine Zeile im sonstigen Programm ändern zu müssen.

3.7 PEEK, POKE und Co. - Systemprogrammierung

Wie man in BASIC programmiert, wissen Sie nun. Im nächsten Kapitel wird es um die Programmierung in Assembler gehen, die bei der Systemprogrammierung eine große Rolle spielt.

Als kleine Überleitung und zur Motivation möchte ich Ihnen deshalb zum Abschluß dieses Kapitels kurz zeigen, wie man in BASIC systemnah programmieren kann.

Der gesamte Speicher des Commodore 64 ist in einzelne Speicherzellen unterteilt, die von 0 bis 65.535 durchnumeriert sind. Jede dieser Speicherzellen kann einen Wert zwischen 0 und 255 aufnehmen.

Der Befehl POKE und die Funktion PEEK

Um den Wert einer einzelnen Speicherzelle zu ändern, gibt es den Befehl POKE:

```
POKE Adresse,Wert
```

Zwei POKE-Befehle, die Sie in der Zwischenzeit sicher schon öfters verwendet haben, sind

```
POKE 53280,FC
```

und

```
POKE 53281,FC
```

Diese beiden Anweisungen ändern die Rahmen- und die Hintergrundfarbe.

```
POKE 53280,0:POKE 53281,0
```

bewirken beispielsweise einen schwarzen Bildschirm. Zum Auslesen des Inhalts einer Speicherzelle benutzt man die Funktion PEEK:

```
PRINT PEEK (53280)
```

beispielsweise bringt den Farbcode der aktuellen Hintergrundfarbe auf den Bildschirm. Während PEEK völlig ungefährlich ist, bringt man den Rechner mit einem falschen POKE sehr schnell zum Absturz. Wie Sie sich vielleicht denken können, verwaltet der Commodore 64 seine sämtlichen internen Abläufe in einer Vielzahl von Speicherzellen.

Wenn Sie nun den Inhalt einer dieser Speicherzellen willkürlich ändern, bringen Sie damit den Commodore 64 unter Umständen völlig aus dem Konzept, und er "verabschiedet" sich. In diesem Fall hilft dann nur noch ein Aus- und Wiedereinschalten des Rechners. Danach ist alles wieder beim alten. Das im Speicher evtl. vorhandene BASIC-Programm wird dabei natürlich gelöscht. Also Vorsicht!

Besonders gefährlich sind die Speicherbereiche 0 bis 2047 und 53248 bis 65535. In diese Bereiche sollte man wirklich nur "hineinpoken", wenn man genau weiß, was man damit anrichtet, wie zum Beispiel beim Ändern der Bildschirmfarben. Ebenfalls sehr interessant ist der sogenannte Bildschirmspeicher von Adresse 1024 bis 2023. In diesem Bildschirmspeicher ist für jede Position des Bildschirms eine Speicherzelle reserviert, in der das Zeichen, das an dieser Position steht, vermerkt wird. Tippen Sie einmal ein:

```
POKE 1024,1:POKE 2023,1
```

In der linken oberen und der rechten unteren Bildschirmecke erscheint ein "A"! Geben Sie nun ein:

```
POKE 1024,1:POKE 1025,2:POKE 1026,3
```

In der ersten Bildschirmzeile erscheint der Schriftzug "ABC". Daran können Sie schon erkennen, wie der Bildschirmspeicher organisiert ist:

```
1024-1063: 1. Bildschirmzeile
1064-1103: 2. Bildschirmzeile
1104-1143: 3. Bildschirmzeile
.....
1984-2023: 25. Bildschirmzeile
```

Wie Sie gesehen haben, hat der Buchstabe "A" den Code 1, der Buchstabe "B" den Code 2 usw... Eine komplette Tabelle dieser sogenannten Bildschirmcodes für die einzelnen Zeichen finden Sie im Anhang. Ein POKE in den Bildschirmspeicher ist übrigens wesentlich schneller als eine entsprechende PRINT-Anweisung. Besonders deutlich wird das, wenn man einen Rahmen um den Bildschirm zeichnen lassen möchte. Versuchen Sie doch einmal, ein Programm zu schreiben, das diese Aufgabe einmal mit PRINT und einmal mit POKE erledigt.

Der Befehl SYS

Bei vielen Programmierproblemen ist es am sinnvollsten, wenn man einen Teil des Programms, das allgemeine Rahmenprogramm, in BASIC und die geschwindigkeitskritischen Teile des Programms in Assembler schreibt. Um nun von BASIC aus ein Maschinenprogramm aufzurufen, gibt es den Befehl SYS. SYS ist in etwa mit GOSUB zum Aufrufen von Unterprogrammen vergleichbar.

Während Sie bei GOSUB die Nummer der Programmzeile, ab der das Unterprogramm beginnt, angeben müssen, folgt hinter SYS die Adresse der Speicherzelle, ab der das Maschinenprogramm im Speicher liegt. Am Ende eines Maschinenprogramms steht, wie bei einem Unterprogramm, ebenfalls ein RETURN, nur schreibt man es in Assembler etwas anders, nämlich RTS.

Auf den Befehl SYS selbst werde ich im nächsten Kapitel noch genauer eingehen. Ich möchte Ihnen hier aber gleich einmal zeigen, was man damit sinnvolles anfangen kann. Das sogenannte Betriebssystem des Commodore 64 besteht aus einer Vielzahl von Maschinenprogrammen (auch Routinen genannt), die man auch für eigene Zwecke nutzen kann. Zum Beispiel die Routine zum Setzen des Bildschirm-Cursors.

Vielleicht haben Sie sich manchmal auch schon gewünscht, einen Text genau an einer bestimmten Stelle des Bildschirms, etwa in der 5. Zeile ab der 10. Spalte, ausgeben zu können. Nichts einfacher als das! Der Commodore 64 vermerkt die momentane Position des Cursors (ab der ja alle über PRINT ausgegebenen

Texte erscheinen) in den Speicherzellen 211 und 214. In Speicherzelle 211 die Spalte, in 214 die Zeile.

Gezählt wird dabei für die Zeile von 0 bis 24, für die Spalte von 0 bis 39. Die Home-Position des Cursors hat also die "Koordinaten" 0/0, während die rechte untere Ecke des Bildschirms mit 24/39 angesprochen wird. Gesetzt wird der Cursor mit einer Betriebssystem-Routine, die ab der Speicherzelle 58.640 beginnt. Wir müssen diese Routine also mit

```
SYS 58640
```

aufrufen. Das komplette Programm sieht dann so aus:

```
10 ze= 5:rem zeile (0 bis 24)
20 sp=10:rem spalte (0 bis 39)
30 poke 211,sp:rem position
40 poke 214,ze:rem setzen
50 sys 58640:rem maschinenprogramm aufrufen
60 print "beispiel"
```

Der Text "BEISPIEL" wird in der 4. Zeile ab der 9. Spalte ausgegeben. Das Setzen des Cursors war nur ein sehr kleines Beispiel für das, was sich mit Maschinensprache erreichen läßt. Sie sehen, es lohnt sich, sich mit Maschinensprache zu befassen!

Der Befehl WAIT

WAIT ist ein Befehl der ganz besonderen Art, nicht zuletzt, weil man ihn in Programmen kaum benötigt. Wie der Name vermuten läßt, dient WAIT dazu, auf etwas zu warten. Der Programmablauf wird durch WAIT so lange angehalten, bis ein bestimmtes Ereignis eintritt.

Dieses "Ereignis" muß in einer Speicherzelle des Commodore 64 stattfinden. WAIT wartet also, bis sich der Inhalt einer Speicherzelle ändert. Sehen wir uns das Format des Befehls einmal genauer an:

```
WAIT Adresse,W1,W2
```

Hinter WAIT steht die Adresse einer beliebigen Speicherzelle (0 bis 65535). Dahinter folgen zwei numerische Werte. Den zweiten Wert kann man auch weglassen; er wird dann automatisch auf Null gesetzt. Schön und gut. Doch was soll das ganze?

Die beiden Werte W1 und W2 werden mit dem momentanen Inhalt der Speicherzelle logisch verknüpft. Zuerst wird der Inhalt mit W2 logisch-EXCLUSIV-ODER verknüpft. Das Ergebnis dieser Verknüpfung wird dann anschließend noch mit W1 logisch-UND verknüpft.

Zum Schluß überprüft WAIT das Gesamtergebnis der beiden Verknüpfungen. Ist es ungleich Null, wird bzw. bleibt der Programmablauf unterbrochen. Ist das Ergebnis gleich Null, wird mit der Programmabarbeitung fortgefahren.

WAIT führt die beiden Verknüpfungen also nicht nur einmal durch, sondern immer wieder - so lange, bis das Ergebnis eben Null ist. Auch wenn Sie es vielleicht kaum für möglich halten, mit WAIT läßt sich durchaus etwas sinnvolles anfangen!

Nehmen wir einmal an, sie möchten in einem Programm auf einen Tastendruck warten. Normalerweise würden Sie das so programmieren:

```
10 get eg$:if eg$="" then 10
```

Wenn man weiß, daß der Commodore 64 die Anzahl der zuletzt gedrückten Tasten in der Speicherzelle 198 vermerkt, geht es zusammen mit WAIT etwas einfacher:

```
10 poke 198,0:wait 198,1
```

Zunächst wird mit POKE 198,0 die Anzahl der gedrückten Tasten auf Null gesetzt. Anschließend wartet WAIT 198,1 bis die Speicherzelle 198 den Wert 1 enthält, d.h., bis eine Taste gedrückt wurde!

BASIC & Betriebssystem

Das Betriebssystem und auch das BASIC stellen viele nützliche Funktionen zur Verfügung. Oft ist es jedoch wünschenswert, diese Funktionen (z.B. LIST) zu beeinflussen, um bestimmte Zwecke zu verfolgen. Von diesen Manipulationsmöglichkeiten soll auf den folgenden Seiten die Rede sein.

Erzeugen von BASIC-Zeilen per Programm

Stellen Sie sich vor, Sie wollten ein Programm schreiben, das den Grafen einer beliebigen Funktion in Hochauflösung auf den Bildschirm zeichnet. Wenn das Programm die Funktion nicht fest vorgeben soll, muß es eine Möglichkeit geben, den Term einzutippen. Für einfachere Versionen reicht es, wenn der Benutzer vorher noch in einer speziellen Programmzeile die Funktion per DEFFN selbst ins Programm einbaut. Doch dazu braucht der Benutzer Programmierkenntnisse. Bequemer wäre es, die Rechenvorschrift über INPUT einzugeben. Doch was nützt uns ein String, in dem ein Term gespeichert ist - ausgeführt werden kann er nicht. Die letzte Möglichkeit wäre, den Rechner sich selbst programmieren zu lassen. Das geht sogar sehr einfach.

Um die Methode zu verstehen, sollten wir zunächst einen Blick auf die normale Entstehung einer Programmzeile werfen. Alles beginnt damit, daß ein Anwender eine (hoffentlich) durchdachte Folge von Buchstaben und Zeichen eintippt. Diese Zeichen erscheinen gleichzeitig auf dem Bildschirm. War eines dieser Zeichen ein RETURN, so übernimmt der BASIC-Interpreter die gesamte Bildschirmzeile (nicht nur die eingetippten Zeichen) in den BASIC-Eingabepuffer und wandelt die Zeichenfolge in eine Programmzeile oder (wenn keine Zeilennummer am Anfang stand) in direkt ausführbare Befehle um. Dem Interpreter ist es also egal, ob die Zeichen eingetipppt oder etwa geprintet wurden. Darauf baut unsere Methode auf. Zunächst wird der beabsichtigte Text der Programmzeile auf dem Bildschirm ausgegeben. Dann müssen wir nur noch die Umwandlung in eine Programmzeile veranlassen. Dazu wird ein künstlicher Tastendruck erzeugt, indem der ASCII-Code in den Tastaturpuffer gepoket wird. Folgt jetzt im Programm ein END, so werden diese Ta-

stendrücke nach dem Programmabbruch ausgeführt. Dabei ergeben sich zwei Probleme. Durch die Erzeugung einer neuen Zeile werden die Variablen gelöscht (wie auch bei der normalen Programmeingabe). Daraus folgt, daß die Erzeugung künstlicher Zeilen erfolgen sollte, wenn keine wichtigen Daten angefallen sind, also am Programmanfang. Müssen einige Variablen erhalten werden, so empfiehlt es sich, diese in freie RAM-Bereiche einzupoken, die vom Betriebssystem nicht benutzt werden.

Zusätzlich soll das Programm nach der Zeilenerzeugung weiterlaufen. Daher muß nach der Zeile ein künstliches GOTO xxx stehen, das nach dem gleichen Muster wie die Zeile erzeugt wird. Hier ein Beispiel-Listing:

```
10 INPUT "Term: Y="; A$: REM Eingabe Funktionsterm
20 PRINT "(CLS)(3xCRSR DOWN)100 DEFFNF(X)="; A$: REM Zeile ausgeben
30 PRINT "GOTO 70(HOME)";: REM Befehl zur Programmfortsetzung
40 POKE 631, 13: POKE 632, 13: REM 2 x RETURN
50 POKE 198, 2: REM Tastaturpuffer initialisieren
60 END
70 ...
```

Wenn Sie dieses Programm eingetippt und gestartet haben, werden Sie sehr schnell den Sinn der einzelnen Anweisungen verstehen, vor allem was die Bildschirmausgabe betrifft. Die erzeugte Zeile unterscheidet sich nicht von einer normal eingegebenen Zeile. Das Programm kann beliebig oft durchlaufen werden. Sollte eine fehlerhafte Eingabe gemacht worden sein, so quittiert der Interpreter dies mit einem SYNTAX-ERROR nach dem Durchlauf der neuen Zeile.

Diese Anwendung läßt sich übrigens noch stark erweitern. So können auf diese Weise Programmzeilen gelöscht werden, die man nicht mehr benötigt. Auch können mehrere Zeilen gleichzeitig erzeugt werden. So ist auch die Eingabe ganzer Unterprogramme per INPUT möglich.

LIST-Schutz

Bei Programmen, die auf persönliche Daten zugreifen, empfiehlt es sich, eine Codewort-Abfrage einzubauen. Damit das Codewort nicht durch LIST aufgedeckt werden kann, sollte die betreffende

Programmzeile geschützt werden. Dies kann beispielsweise durch einen POKE-Befehl erreicht werden.

Zum Verständnis ist es nötig, das Format einer Programmzeile im Speicher zu kennen. Die ersten beiden Bytes einer Zeile bilden den Zeiger auf die nächste Zeile. Damit kann sich der Interpreter von Zeile zu Zeile "hangeln". Sind diese beiden Bytes 0, so ist danach keine Programmzeile mehr gespeichert; hier befindet sich also das Programmende.

Nach dem Zeiger folgen zwei Bytes mit der Zeilennummer. Auch diese ist wie ein Pointer aufgebaut. Dann folgen die Befehle im Interpretercode. Das Zeilenende wird durch eine Null repräsentiert. Mit dieser 0 können wir den Interpreter ein wenig hereinlegen. POKEn wir nämlich direkt nach der Zeilennummer eine 0 ein, so meint die LIST-Routine, die Zeile wäre bereits abgeschlossen und holt sich die nächste Programmzeile (der Pointer am Zeilenanfang blieb ja unverändert). Auch ein GOTO wird dadurch nicht beeinflußt, da die Routine, die eine bestimmte Zeile im Text sucht, sich ebenfalls an diesen Pointern orientiert. Die Routine, die den nächsten Befehl im Programm sucht, tut dies aber nicht, sondern überspringt nach einer 0 einfach 4 Bytes. Deshalb "fehlen" die ersten vier Bytes der Zeile beim Programmablauf. Um die Ausführung der Befehle nicht zu behindern, müssen beim Schreiben der Programmzeile 5 beliebige Zeichen (aber kein Befehlswort) eingefügt werden. Das erste dieser Zeichen wird durch die 0 überschrieben, die restlichen vier dienen als Platzhalter.

Woher wissen wir aber, welches Byte wir überschreiben müssen? Nun, auch dafür gibt es einen Trick. Wir bauen vor der zu schützenden Zeile einen STOP-Befehl ein und lassen das Programm bis hierhin ablaufen. Nach dem BREAK steht in den Speicherzellen 61 und 62 der Pointer auf dem nächsten BASIC-Befehl. Wenn der STOP-Befehl am Ende der Zeile steht, zeigt der Pointer auf das Zeilenende, also auf eine Null. Addiert man zu dieser Adresse noch 5 dazu, so erhält man das gewünschte Byte. Also frisch ans Werk mit POKE AD, 0. Nach diesem Befehl erscheint beim "listen" nur noch die Zeilennummer, der Text wird nicht mehr gezeigt. Es bleibt nur noch, den STOP-

abgelegt, die Adresse des Bytes nach dem letzten DATA-Element, das gelesen wurde, finden wir in 65/66.

Wollen wir jetzt ein RESTORE simulieren, so können wir folgendermaßen vorgehen:

1. DATAs bis zum Element vor dem gewünschten Ziel lesen lassen (z.B. im Direktmodus). Soll auf das 5. Element zurückgesetzt werden, so müssen also die ersten 4 DATAs gelesen werden.

2. PRINT PEEK (63), PEEK (64)
   Die erscheinenden Zahlen repräsentieren die Zeilennummer. Zahlen bitte merken!

3. PRINT PEEK (65), PEEK (66)
   Auch diese Zahlen müssen wir uns merken! Sie bilden den Zeiger auf das Byte nach dem letzten DATA-Element. Bis hierhin müssen alle Befehle vor dem eigentlichen Programmablauf gegeben werden.

4. POKE 63, 1. Zahl: POKE 64, 2. Zahl
   POKE 65, 3. Zahl: POKE 66, 4. Zahl

Diese Befehle werden statt RESTORE ins Programm an die Stelle eingebaut, an der der Datazeiger zurückgesetzt werden soll. Dadurch werden die Pointer auf den Stand gebracht, den Sie vor dem Lesen des gewünschten Elements hatten. Für das BASIC entsteht der Eindruck, als hätte es die nachfolgenden DATA-Zeilen noch nicht gelesen.

Allerdings hat diese Methode einen Nachteil. Nach jeder Änderung in Programmzeilen, die vor der gewünschten Position des Zeigers liegen, ändert sich die Adresse, die im DATA-Pointer stehen sollte, da das BASIC den gesamten Programmtext im Speicher verschiebt. Deshalb sollten solche DATA-Blöcke ganz am Anfang des Programms vor den eigentlichen Befehlen stehen.

Zusammenfassung: RESTORE

Zeilennummer des letzten DATA-Elements ist in den Speicherzellen 63 und 64 gespeichert. Die Adresse des Bytes nach dem letzten Element befindet sich als Zeiger in den Bytes 65 und 66. Beide Pointer können durch POKE verändert werden (vorher gewünschte Pointer-Werte feststellen).

Verschiedene Tricks

Nach einer Programmunterbrechung oder einem ERROR zeigt der Rechner an, in welcher Zeile das Programm verlassen wurde. Hat man etwas voreilig den Bildschirm gelöscht, so erfährt man diese Zeilennummer meist nicht mehr. Hier schaffen die Speicherzellen 59 und 60 Abhilfe. Hier wird (im Zeigerformat) die letzte Zeilennummer abgelegt, die man sich durch

```
PRINT PEEK (59) + 256 * PEEK (60)
```

ausgeben lassen kann. Ein Programm vor SAVE schützen kann man mit dieser Sequenz:

```
POKE 801, 0: POKE 802, 0: POKE 818, 165
```

Dadurch werden die Vektoren, die der SAVE-Befehl benötigt, so umgebogen, daß kein Abspeichern mehr möglich ist. Nachteil: Schon durch einfaches Drücken von <Run/Stop>-<Restore> hängt sich der Rechner auf. Schließlich noch einige SYS-Befehle, die sich gut in eigenen Programmen verwenden lassen:

```
SYS 65499
```

setzt den TI$ auf 000000. Das geht schneller als die Zuweisung eines neuen Strings. Ästheten unter den Commodore-Besitzern können ein Programm durch SYS 42115 (statt END) beenden. Damit wird ein Warmstart des BASIC bewirkt, was nichts anderes heißt, als daß das BASIC in den Direktmodus umschaltet. Dabei wird aber kein READY ausgegeben; der Cursor steht sofort in der nächsten Zeile. Auch ein CONT bleibt nach SYS erfolglos.

Soll das Programm mit dem Einschaltbild beendet und gleichzeitig gelöscht werden, so bietet sich SYS 58253 an. Und einen künstlichen SYNTAX ERROR erzielt man durch SYS 44808.

Zusammenfassung: Tricks zum Betriebssystem

Letzte Zeilennummer ist in Speicherzellen 59 und 60 gespeichert.

```
SAVE-Schutz: POKE 801, 0: POKE 802, 0: POKE 818, 165
TI$ auf 0 setzen: SYS 65499
End ohne Ready: SYS 42115
Einschaltbild: SYS 58253
Syntax Error: SYS 44808
```

BASIC-Erweiterungen

Fast jeder Commodore-Besitzer kennt BASIC-Erweiterungen zumindest aus Anzeigen, wenn er nicht sogar selbst ein solches Programm besitzt. Die ersten Exemplare dieser nützlichen Helfer gab es schon zu den Zeiten des seligen PET 2001. Zunächst enthielten sie nur sogenannte Toolkit-Befehle, die das Editieren von Programmen erleichterten. Darunter fällt zum Beispiel auch AUTO. Dieser Befehl gibt automatisch die Zeilennummern im gewählten Abstand (z.B. 10) für die einzugebenden Programmzeilen vor, so daß man sich diese Tipparbeit sparen kann. FIND findet bestimmte Ausdrücke im Programmtext, RENUMBER, MERGE und RENEW kennen Sie bereits. Mit DEL können Sie ganze Programmteile löschen. TRACE ermöglicht durch Ausgabe der durchlaufenen Zeilen eine einfache Überwachung des Programmablaufs beim Testen. DUMP gibt alle benutzten Variablen samt Inhalt aus.

Komfortablere Versionen ermöglichen auch das Auffangen von ERRORS. Damit wird so zum Beispiel die Korrektur von fehlerhaften Eingaben ermöglicht, ohne daß ein TYPE-MISMATCH-ERROR erscheint.

Seltener findet man noch die Möglichkeit, die Funktionstasten mit Zeichenfolgen zu belegen.

Da das BASIC des 64ers die phantastischen Sound- und Grafikmöglichkeiten nicht unterstützt, bieten viele BASIC-Erweiterungen auch hier Befehle zum Zeichnen und zum Programmieren von Tonfolgen. Einige Programme stellen auch Strukturierungsbefehle zur Verfügung, mit denen man Programme ohne GOTO-Befehle schreiben kann. In diesem Fall werden die einzelnen Programmteile in sogenannten Moduln (ähnlich Unterprogrammen) programmiert. Statt GOSUB wird jetzt z.B. mit CALL PLOT X,Y aufgerufen, um eine Punkt-Setzroutine zu erreichen. Diese Technik fördert die Übersichtlichkeit eines Programms sehr. Verbreitet ist auch der Einbau von speziellen DOS-Befehlen, die es z.B. ermöglichen, eine Directory direkt auf den Bildschirm zu holen, ohne ein Programm im Speicher zu löschen.

Wenn etwas schiefgeht: Unser Pannen-Service

Weil Autoren nicht fehlerlos sind und Druckfehler sowieso außerhalb des Einflusses der Autoren liegen, finden Sie in diesem Kapitel einige allgemeine Erste-Hilfe-Regeln, in der Hoffnung, daß dann nichts mehr schiefgehen kann.

Kommen wir jetzt aber zu speziellen Fehlerquellen.

Panne: Programmänderungen werden nicht durchgeführt.

Sie haben in einem Programm eine Zeile geändert, aber anschließend funktioniert das Programm nicht entsprechend der Änderung.

Lösung: Wahrscheinlich haben Sie das Drücken der <Return>-Taste vergessen. Lassen Sie sich mit LIST die geänderte Zeile zeigen, nehmen Sie die Änderung erneut vor, und betätigen Sie zum Abschluß die <Return>-Taste.

Panne: Cursor produziert Grafikzeichen.

Sie wollen in einer Zeile eine Änderung vornehmen, aber der Cursor funktioniert nicht mehr wie normal, sondern produziert seltsame Grafikzeichen.

Lösung: Sie haben mit der <Inst>-Taste Platz geschaffen oder ein Anführungszeichen verwendet. Dadurch befindet sich der C64 im sogenannten "Einfügemodus", bei dem die Cursor-Tasten eine andere Funktion haben. Drücken Sie <Shift>+<Return>, dadurch verläßt der Cursor die Zeile, ohne diese auszuführen, und Sie können erneut in die Zeile gehen und den Cursor dort normal bewegen.

Panne: PRINT ergibt statt eines Wortes eine 0.

Sie wollen ein Wort auf den Bildschirm schreiben, aber der PRINT-Befehl gibt nicht das gewünschte Wort aus, sondern eine 0.

Lösung: Sie haben die Anführungszeichen vergessen. Wenn Sie etwas unverändert und ohne irgendeine Berechnung auf den Bildschirm ausgeben wollen, so muß es in Anführungszeichen stehen.

Panne: PRINT ergibt statt Variablenwert immer Variablennamen.

Der PRINT-Befehl gibt nicht den Wert der Variablen, sondern den Variablennamen aus. Sie haben beispielsweise in A$ den Namen "Peter" gespeichert, erhalten aber beim PRINT-Befehl immer "A$" auf dem Bildschirm.

Lösung: Versehentlich haben Sie ein Anführungszeichen gesetzt. Dadurch gibt BASIC das folgende ohne jede Änderung auf dem Bildschirm aus. Entfernen Sie das Anführungszeichen.

Panne: BASIC reagiert nicht auf Befehle im Direktmodus.

Sie wollen im Direktmodus (also ohne Zeilennummer) einen Befehl ausführen lassen. BASIC führt diesen Befehl aber nicht aus, sondern setzt den Cursor nach Betätigung der <Return>-Taste einfach eine Zeile tiefer. Beispielsweise befehlen Sie

```
RUN
```

aber das Programm startet nicht.

Lösung: Wenn Sie richtig die <Return>-Taste betätigt haben und zu Beginn der Zeile auch wirklich keine Zahl steht (sonst Programm-Modus), dann gibt es nur eine Erklärung: Sie geben den Befehl RUN in einer verlängerten Programmzeile ein. BASIC kann nämlich auch mit Programmzeilen arbeiten, die zwei Bildschirmzeilen lang sind. Das passiert immer dann, wenn Sie bei der Eingabe einer Programmzeile über den rechten Bildschirmrand hinausschreiben. Selbst wenn Sie mit der Backspace-Taste die Zeichen in der zweiten Zeile löschen, gehört diese zweite Zeile immer noch als Programmzeile zur ersten. Machen Sie zweierlei:

1. Merken Sie sich die Nummer der Programmzeile, bewegen Sie den Cursor mit den Cursor-Tasten in die unterste Bildschirmzeile, und drücken Sie einige Male die <Return>-Taste. Nun sind Sie sicherlich nicht mehr in einer erweiterten Programmzeile.

2. Lassen Sie sich mit LIST die gemerkte Zeile zeigen, und kontrollieren Sie, ob diese noch Reste Ihrer Befehlseingaben enthält - beispielsweise noch das RUN. Entfernen Sie diese Reste gegebenenfalls. Wenn Sie nach der Änderung <Return> drücken, springt der Cursor zwei Zeilen tiefer. Dort können Sie den gewünschten Befehl eingeben.

Noch ein Tip zu erweiterten Programmzeilen. Diese erkennt man eben daran, daß der Cursor beim Druck auf die Eingabetaste zwei Zeilen weiter nach unten springt.

Panne: Unerklärlicher SYNTAX ERROR im Direktmodus.

Sie geben im Direktmodus (also ohne Zeilennummer) einen korrekten Befehl ein, aber BASIC meldet einen SYNTAX ERROR oder eine andere Fehlermeldung, die Sie sich nicht erklären können.

Lösung: Wahrscheinlich befinden sich in der Zeile, in der Sie Ihre Eingabe machen, noch irgendwelche Zeichen, beispielsweise von einer vorherigen Bildschirmausgabe. Geben Sie einfach in der Zeile den gewünschten Befehl mit einem abschließenden Doppelpunkt ein, oder setzen Sie den Cursor in eine Zeile, in der keine Reste einer Bildschirmausgabe mehr vorhanden sind.

Panne: ?REDO FROM START beim INPUT-Befehl.

Sie wollen in einem laufenden Programm hinter dem Fragezeichen eine Eingabe machen. Wenn Sie aber etwas eingeben, arbeitet das Programm nicht weiter, sondern es erscheint immer die Meldung:

```
?REDO FROM START
```

auf dem Bildschirm.

Lösung: Beim INPUT-Befehl muß ja stets eine Variable angegeben werden, die die Eingabe aufnehmen soll. In Ihrem Fall ist es eine numerische Variable, die nur Zahlen enthalten darf, Sie geben aber einen String (Zeichenkette) ein. Entweder haben Sie beim Variablennamen das $-Zeichen vergessen, oder Ihre Eingabe enthält Buchstaben und Sonderzeichen.

Panne: Scheinbar korrekte Programmzeile ergibt SYNTAX ERROR

Sie haben eine Programmzeile mit Variablen und starten das Programm. Der C64 meldet immer einen SYNTAX ERROR in der Zeile, obwohl diese völlig in Ordnung zu sein scheint.

Lösung: Sie haben versehentlich einen Befehl oder ein reserviertes Wort im Variablennamen verwendet, zum Beispiel ON in TELEFON. Benutzen Sie einen anderen Variablennamen. Kurze Variablennamen sind zwar weniger ausdrucksstark, aber verhindern dieses Problem. Schauen Sie im Lexikon unter Variablennamen nach.

Panne: Offensichtlich unsinnige Werte bei Berechnungen.

Sie wollen in einem Programm etwas berechnen lassen, doch der C64 kommt immer zu sehr eigenartigen Ergebnissen. Sie haben schon mehrfach mit dem Taschenrechner nachgerechnet und glauben eher diesem als dem Programm.

Beispiel:

```
10 KILO = 10
20 KILOMETER = 20
30 PRINT KILO/KILOMETER
```

Dies Programm ergibt als Ausgabe eine "1", obwohl eigentlich 0.5 herauskommen müßte.

Lösung: Die beiden Variablennamen unterscheiden sich für den C64 nicht, da er nur die ersten beiden Zeichen berücksichtigt. Für ihn steht dort:

```
10 KI = 10
20 KI = 20
30 PRINT KI/KI
```

Wählen Sie Variablennamen, die sich schon in den ersten beiden Zeichen unterscheiden, beispielsweise KI für KILO und KM für KILOMETER.

Panne: Programm versehentlich gelöscht.

Sie haben ein kleines Programm geschrieben und nicht gespeichert.

Nun haben Sie versehentlich NEW eingegeben oder durch das Laden des Inhaltsverzeichnisses das Programm im Speicher gelöscht. Damit ist das Programm leider verloren.

Lösung: Trotzdem können Sie all das, was noch auf dem Bildschirm sichtbar ist, leicht wiederholen, indem Sie einfach in jede noch sichtbare Programmzeile gehen und auf die <Return>-Taste drücken. Achtung: Im Falle des Inhaltsverzeichnisses müssen Sie vorher NEW eingeben, sonst bleiben Reste davon im Speicher.

Panne: Sie starten ein gerade neu geschriebenes Programm mit RUN und erhalten einen SYNTAX ERROR IN 0, obwohl Sie natürlich wie immer mit Zeile 10 begonnen haben.

Halt: .....

Lösung: Denken Sie einen Augenblick nach. Haben Sie vor dem Schreiben des Programms das Inhaltsverzeichnis der Diskette geladen und das anschließende NEW vergessen? Dann geben Sie jetzt kein LIST ein. Sie können den noch auf dem Bildschirm sichtbaren Teil des Programms retten, indem Sie NEW eingeben und anschließend alle Zeilen auf dem Bildschirm durch <Return> übernehmen.

Panne: TYPE MISMATCH ERROR in ...

Sie erhalten die Fehlermeldung: TYPE MISMATCH ERROR in 90.

Lösung: Dieser Fehler tritt auf, wenn Sie numerische Variablen mit String-Variablen verwechseln oder gleichsetzen. Numerische Variablen können nur Werte von anderen numerischen Variablen zugewiesen bekommen, Zahlen oder das Ergebnis von Berechnungen aufnehmen. String-Variablen können nur Zeichenketten (in Anführungszeichen) oder den Inhalt anderer String-Variablen aufnehmen. So sind beispielsweise die folgenden Zuweisungen nicht erlaubt:

```
Eingabe = "Hallo" (Links numerisch, rechts String)
Eingabe$ = 4 (Links String, rechts numerisch)
Eingabe$ = Meineeingabe (Links String, rechts numerisch)
```

Panne: Programm will nicht stoppen.

Sie haben ein Programm geschrieben, das nach dem Start gar nicht mehr aufhören will, weil es eine Endlosschleife enthält.

Lösung: Drücken Sie <Run/Stop>, und BASIC stoppt das Programm. Nun sollten Sie natürlich vor einem neuen Start erst einmal das Programm so abändern, daß es auch normal beendet werden kann. Manchmal funktioniert <Run/Stop> nicht, weil das

Programm INPUT-Befehle enthält, und während der C64 das INPUT abarbeitet, reagiert er nicht auf <Run/Stop>. Drücken Sie in einem solchen Fall gleichzeitig <Run/Stop> + <Restore>.

Panne: NEXT WITHOUT FOR ERROR IN ...

Sie erhalten die Fehlermeldung NEXT WITHOUT FOR ERROR IN 30.

Lösung: Hier können mehrere Fehler vorliegen. Haben Sie überhaupt vor der Zeile 40 eine Zeile, die die Schleife beginnt? Wenn ja, so haben Sie sich vielleicht mit dem Variablennamen für die Schleife verschrieben. Zu einen "For I = 1 to 100" gehört auch ein "Next I" und kein "Next J". Eventuell verwenden Sie auch eine geschachtelte Konstruktion aus zwei Schleifen. Dann muß die zuletzt begonnene Schleife zuerst wieder beendet werden. Beispielsweise ist folgende verschachtelte Schleife nicht richtig:

```
10 For I = 1 to 10
20 For J = 1 to 10
30 Print I,J
40 Next I Rem (Hier muß erst die innere Schleife (J)
                    geschlossen werden )
50 Next J
```

Panne: RETURN WITHOUT GOSUB ERROR IN ...

Sie erhalten die Fehlermeldung RETURN WITHOUT GOSUB ERROR IN 30.

Lösung: Wenn BASIC auf ein RETURN trifft, so will es zu der Zeile zurückkehren, in der das zugehörige GOSUB gestanden hat. In diesem Fall ist BASIC aber auf ein RETURN gestoßen, ohne daß vorher GOSUB befohlen wurde. Haben Sie das GOSUB vergessen? Wenn nein, so gibt es noch eine häufige Fehlerquelle: Sie haben ein Hauptprogramm, das ein Unterprogramm aufruft. Das Hauptprogramm endet aber nicht mit einem END, und daher "läuft" BASIC irgendwann fälschlicherweise in das Unterprogramm. Dieser Fehler tritt beispielsweise in folgendem Programm auf:

```
10 REM Dies ist das Hauptprogramm
20 FOR I = 1 TO 100
30 GOSUB 100
40 NEXT I

100 REM Unterprogramm
110 PRINT I
120 RETURN
```

Fügen Sie die folgende Zeile ein:

```
50 END
```

Panne: UNDEF'D STATEMENT ERROR in ...

Sie erhalten beispielsweise die Fehlermeldung UNDEF'D STATEMENT ERROR in 30.

Lösung: Sie haben versucht, das Programm mit GOTO springen zu lassen, aber die angegebene Zeilennummer existiert gar nicht. Kontrollieren Sie auf Tippfehler, und schauen Sie nach, ob Sie die Zielzeile vielleicht gelöscht haben.

Panne: BAD SUBSCRIPT ERROR in ...

Sie erhalten nach dem Starten des Programms die Fehlermeldung: BAD SUBSCRIPT ERROR in 20.

Lösung: Sie arbeiten mit einem Feld und versuchen ein Element dieses Feldes anzusprechen, das außerhalb des durch DIM angegebenen Bereiches liegt. Beispiel:

```
10 DIM A(100)
20 A(200) = 4
```

Panne: Überraschender OUT OF DATA ERROR.

Völlig überraschend taucht nach Betätigung der <Return>-Taste obige Fehlermeldung auf, obwohl Sie in Ihrem Programm weder DATA-Zeilen noch den READ-Befehl verwenden.

Lösung: Immer, wenn der C64 mit etwas fertig ist, meldet er sich mit einem READY. Schauen Sie sich das READY noch

einmal genau an. Der C64 interpretiert diese Zeile als READ Y. Sie haben versehentlich in dieser Zeile die <Return>-Taste betätigt und haben keine DATA-Zeilen, aus denen der C64 sein Y holen könnte.

Panne: FILE NOT FOUND ERROR

Sie wollen ein Programm oder das Inhaltsverzeichnis der Diskette laden und erhalten diese Fehlermeldung. Zusätzlich blinkt die rote Lampe der Floppy.

Lösung: Sie haben entweder keine Diskette eingelegt oder den Diskettenschacht nicht geschlossen, oder das Programm befindet sich nicht auf der Diskette (falsche Schreibweise). Beim nächsten richtigen Zugriff auf die Floppy hört das Blinken auf.

Panne: DEVICE NOT PRESENT ERROR

Sie wollen das Diskettenlaufwerk benutzen und erhalten diese Fehlermeldung.

Lösung: Sie haben vergessen, die Floppy einzuschalten. Sollte sich nach dem Einschalten nichts ändern, so schalten Sie alle Geräte aus und überprüfen die Verbindungskabel. Achtung: Nur im ausgeschalteten Zustand Verbindungskabel herausziehen oder einstecken.

Panne: Floppylampe blinkt beim Speichern.

Sie speichern ein Programm, und der C64 meldet auch die Ausführung mit READY, aber die Lampe der Floppy blinkt.

Lösung: Sie haben versucht, ein bereits unter demselben Namen bestehendes Programm noch einmal abzuspeichern. Löschen Sie entweder das alte Programm, oder verwenden Sie den Klammeraffen oder aber wählen Sie einen anderen Namen.

Panne: Speichern ohne Erfolg.

Sie haben ein Programm in einer neuen Version gespeichert. Beim späteren Laden entpuppt es sich als eine ältere Version.

Lösung: Sie haben es versehentlich unter demselben Namen zu speichern versucht und nicht gemerkt, daß die Floppylampe blinkte, weil Sie anschließend sofort das Inhaltsverzeichnis geladen haben.

4. Auch Assembler ist nicht schwer

Nachdem wir uns im letzten Kapitel ausführlich mit BASIC befaßt haben, möchte ich Ihnen nun zeigen, wie man dem Commodore 64 in Maschinensprache bzw. Assembler zu Leibe rückt. Auch wenn Sie im Augenblick nocht nicht in die Assembler-Programmierung einsteigen möchten, sollten Sie dieses Kapitel nicht einfach überblättern. Unterkapitel 4.7 enthält nämlich eine Reihe hochinteressanter Routinen, die auch für alle Nur-BASIC-Programmierer von Nutzen sind.

4.1 Warum überhaupt Assembler?

Sicherlich haben Sie in der Zwischenzeit fleißig in BASIC programmiert und festgestellt, daß sich damit viele Aufgaben auf leichte Art und Weise lösen lassen. Warum soll man sich da noch mit Assembler befassen? Ist dies nicht nur noch ein Relikt aus den Anfangstagen der Computerei? Vergleichen wir deshalb einmal BASIC mit Assembler.

Daß BASIC nicht schwer zu erlernen ist, haben Sie im letzten Kapitel gesehen. Lassen Sie sich aber in diesem Kapitel überzeugen, daß das Programmieren in Assembler ebenso einfach und schnell zu erlernen ist. Dabei kommt Ihnen zugute, daß Sie bereits BASIC beherrschen. Die grundsätzliche Arbeitsweise ist bei der Maschinensprache nicht viel anders. Welche Vorteile gegenüber BASIC lassen es nun gerechtfertigt erscheinen, sich eine neue Programmiersprache anzueignen?

BASIC zählt zu den sogenannten höheren Programmiersprachen wie zum Beispiel auch FORTRAN, PASCAL oder COBOL. Diese Sprachen werden oft auch als problemorientierte Sprachen bezeichnet, da sie sich an den zu lösenden Problemen, z.B. mathematischen Berechnungen oder kommerziellen Anwendungen, orientieren. Dem gegenüber gibt es sogenannte maschinenorientierte Sprachen, wie z.B. FORTH, die sich an der Hardware des

Rechners orientieren. Dazu gehört als Extremfall natürlich auch die Maschinensprache des Prozessors selbst.

BASIC verstehen Sie also gut, sonst würden Sie sich jetzt wohl kaum an die Programmierung in Maschinensprache herantrauen. Da sieht es mit Ihrem Commodore 64 schon ganz anders aus. Der versteht nämlich BASIC überhaupt nicht. Wie kommt es dann, so werden Sie zurecht fragen, daß er BASIC-Befehle dann so bereitwillig und schnell ausführt? Dafür sorgt der im Betriebssystem Ihres Commodore 64 enthaltene BASIC-Interpreter. Dieser Interpreter präsentiert Ihnen BASIC im Klartext. Geht es jedoch an die Ausführung der von Ihnen geschriebenen Programme, so muß jeder einzelne Befehl von Ihrem Commodore 64 erst interpretiert werden, d.h., in entsprechende, von ihm ausführbare Einzelschritte übertragen werden. Das braucht Sie eigentlich nicht weiter zu stören, denn schließlich erledigt der Interpreter seine Aufgabe doch souverän und für Sie kaum merkbar.

Schauen wir uns an einem einfachen Beispiel an, wie die Arbeit des Interpreters vor sich geht:

```
PRINT "HALLO"
```

Wenn Sie diesen Befehl eingeben und <Return> drücken, wird der Befehl Zeichen für Zeichen vom Interpreter gelesen. Sobald er das erste Wort gelesen hat, geht der Interpreter hin, vergleicht dieses Wort mit allen Worten aus seiner Befehlstabelle (z.B. GOTO, FOR, INPUT usw...) und prüft, ob es in der Tabelle enthalten ist. Hat er es gefunden, so merkt er sich, an wievielter Stelle dieses Wort in der Tabelle steht. Diese Position braucht der Interpreter, um festzustellen, an welcher Stelle innerhalb des Interpreters der PRINT-Befehl steht. Nun kann der eigentliche PRINT-Befehl ausgeführt werden. Auch hier wird wieder zeichenweise gelesen. An den Anführungszeichen merkt der Interpreter, daß Sie einen Text drucken wollen. Er gibt Zeichen für Zeichen aus, bis er mit dem nächsten Anführungszeichen das Ende des Wortes entdeckt. Nun prüft er, ob noch weiterer Text folgt. Ist das nicht der Fall, so ist der Befehl ausgeführt und der Interpreter meldet sich mit READY wieder.

Sicherlich werden Sie jetzt sagen: Das ist aber umständlich, da muß es doch was besseres geben. Stimmt! Und genau deshalb wollen wir uns jetzt einmal ansehen, welche Vorteile es hat, direkt in Maschinensprache zu programmieren.

Maschinensprache ist erheblich schneller! Welchen Geschwindigkeitsvorteil bringt die Maschinensprache gegenüber BASIC und warum ist das so? Damit Ihr Rechner BASIC überhaupt versteht, muß er einen sogenannten BASIC-Interpreter haben, der die einzelnen BASIC-Befehle erkennt und ausführt. Dieser Interpreter ist selbst in Maschinensprache geschrieben. Um z.B. den Befehl POKE 1024,10 auszuführen, muß zuerst der Befehl vom Interpreter erkannt werden. Danach können die Argumente geholt und die 10 in die Speicherstelle 1024 geschrieben werden. Das ganze dauert etwa 2 Millisekunden, das sind 2 tausendstel Sekunden. Nicht viel Zeit, könnte man meinen. Wie sieht das ganze jetzt in Maschinensprache aus? Hier sind zwei Maschinenbefehle erforderlich:

```
LDA #10
STA 1024
```

Diese beiden Befehle brauchen nur 6 Mikrosekunden, das sind 6 millionstel Sekunden, das ist nicht einmal der dreihundertste Teil davon!

Man kann davon ausgehen, das ein reines Maschinenprogramm ca. 10 bis 1000 mal schneller ist als ein BASIC-Programm für die gleiche Aufgabe.

Besonders zeitintensive Aufgaben sind z.B. umfangreiche mathematische Berechnungen und besonders auch das Sortieren von Daten. Bei umfangreichen Datenbeständen kann dies in BASIC leicht Stunden dauern. Hier kommt man oft ohne die Maschinensprache nicht mehr aus.

Während man hier mit BASIC lediglich Zeit verschenkt, lassen sich andere Aufgaben ohne Maschinensprache prinzipiell nicht lösen. Dazu gehört zum Beispiel die Programmierung von Ein-/ Ausgabebausteinen zur Übertragung von Daten oder Routinen,

die im "Interrupt" ausgeführt werden, eine Technik, die in BASIC nicht zur Verfügung steht. Interrupt heißt zu deutsch Unterbrechung und bedeutet, daß z.B. Bausteine des Rechners oder externe Geräte die Arbeit des Rechners jederzeit unterbrechen können und den Rechner zwingen, diese Geräte entsprechend zu bedienen.

Generell gilt aber, daß man alle Möglichkeiten seines Rechners nur mit Maschinensprache voll ausschöpfen kann. Dies gilt beim Commodore 64 speziell für die hochauflösende Grafik und den Synthesizer. Dies gilt auch für alle Sachen, die in "Echtzeit" programmiert werden müssen.

Ein weiterer wichtiger Punkt ist die Speicherplatzausnutzung. Ein sehr gut geschriebenes Maschinenprogramm kann durchaus zehnmal kürzer sein, als ein entsprechendes Programm in BASIC. Ein BASIC-Programm von ein KByte Größe kann man nicht als sehr groß bezeichnen, für ein Maschinenprogramm ist dies jedoch schon eine ansehnliche Größe.

Das gleiche gilt auch für die Datenspeicherung. Haben Sie zum Beispiel eine Tabelle, so belegt im günstigsten Fall jedes Feldelement zwei Bytes, wenn Sie Integer-Felder benutzen, obwohl Sie nur Zahlen zwischen Null und Hundert abspeichern müssen. In Maschinensprache ist es kein Problem, hier nur ein Byte pro Element zu benutzen. In Maschinensprache können Sie also die jedem Problem optimal angepaßte Datenstruktur wählen.

Nun müssen wir aber fairerweise auch fragen, welche Nachteile wir uns durch die Maschinensprache-Programmierung einhandeln. Nun - erst müssen wir das Programmieren in Maschinensprache einmal lernen. Wenn Sie jedoch BASIC beherrschen, bringen Sie bereits die Grundvoraussetzungen dafür mit. Ein Nachteil von Maschinenprogrammen ist, daß sie prinzipiell nur auf Maschinen laufen, die den gleichen Prozessortyp enthalten, und auch auf diesen von Gerät zu Gerät größere Anpassungen erforderlich sein können. Dem kann man jedoch schon bei der Programmierung Rechnung tragen. Auch ist das Austesten von Maschinenprogrammen ohne entsprechende Werkzeuge nicht so einfach wie in BASIC.

Zusammenfassend können wir sagen, daß die Maschinenprogrammierung durchaus ihre Berechtigung hat. Viele Aufgaben lassen sich ohne Maschinenprogrammierung nicht lösen, und nur mit ihr kann man "das Letzte" aus seinem Rechner herausholen. Oft wird man seine Grundprogramme auch weiterhin in BASIC schreiben und lediglich Teilprobleme in Maschinensprache formulieren.

Diese Vorgehensweise ist sehr beliebt und begegnet Ihnen dauernd in Form von Programmierhilfen oder BASIC-Erweiterungen, so auch in diesem Buch.

Haben Sie erst Ihr erstes eigenes Maschinenprogramm geschrieben, so werden auch Sie feststellen, daß es gar nicht so schwierig ist.

4.2 Die erforderlichen Werkzeuge

Vielleicht haben Sie sich schon gewundert, daß ich die beiden Begriffe Maschinensprache und Assembler nebeneinander benutze. Worin liegen die Unterschiede? Dazu muß ich etwas ausholen. Die einzige Sprache, die der Commodore 64 wirklich "versteht", ist die Maschinensprache. Darunter können Sie sich lange Folgen von Nullen und Einsen vorstellen, die intern im Rechner in Form von Spannungsimpulsen und Magnetisierungszuständen dargestellt werden. Da solch eine Sprache selbst für hartnäckigste Computerfreaks kaum handhabbar ist, hat man den sogenannten Assembler eingeführt. In dieser Sprache werden die Maschinenbefehle, die man dem Rechner geben kann, nicht mehr durch Nullen und Einsen ausgedrückt, sondern durch symbolische Namen. Ein Befehl zum Laden eines Wertes in eine Speicherzelle heißt dann zum Beispiel nicht mehr "10011101", sondern "STA". Damit läßt sich doch schon wesentlich leichter umgehen!

Der Begriff Assembler meint aber noch etwas anderes. Da der Commodore 64 die Assembler-Befehle ja nicht direkt "versteht", benötigt man ein Werkzeug, das ihm diese in Maschinensprache

übersetzt. Solch ein Übersetzungsprogramm bezeichnet man im allgemeinen ebenfalls als Assembler.

Und damit wären wir auch schon bei den zum Programmieren in Assembler erforderlichen Werkzeugen angelangt. Leider hat der Commodore 64 diese Hilfsmittel, wie den BASIC-Interpreter standardmäßig, nicht eingebaut.

Was Sie unbedingt benötigen, ist natürlich einen Assembler. Zwei weitere sehr hilfreiche Werkzeuge sind ein sogenannter Diassembler sowie ein Einzelschrittsimulator. Einfache, in BASIC programmierte Versionen dieser drei Werkzeuge finden Sie am Ende des Kapitels. Professionelle Programme sind meist in Maschinensprache geschrieben und dadurch natürlich wesentlich schneller und komfortabler. Für Ihre ersten Gehversuche in Maschinensprache reichen die abgedruckten Programme aber völlig aus.

Kommerzielle Assembler-Werkzeuge sind mittlerweile recht günstig zu haben. Wenn Sie langfristig in Assembler programmieren wollen, lohnt sich auf jeden Fall ein Gang ins Fachgeschäft oder ein Blick in den Listing-Teil einer Fachzeitschrift.

Durch die Vielzahl der angebotenen Programme ergibt sich ein weiteres Problem: Wie auch Anwendungsprogramme ihre Funktionen mitunter durch sehr unterschiedliche Aufrufkommandos anbieten, so gibt es auch bei den einzelnen Assemblern zum Teil erhebliche Unterschiede in der Bedienung. Sollte es trotzdem einmal zu Problemen kommen, dürfte ein Blick in die Bedienungsanleitung zu Ihrem Programm für Klarheit sorgen.

4.3 Der 6510-Mikroprozessor

Der Mikroprozessor, kurz die CPU, besitzt einen "Programmzähler" oder "program counter". Der Programmzähler weist immer auf die Adresse, an der sich der nächste zu bearbeitende Befehl befindet.

Da der gesamte Rechnerspeicher addressiert werden muß ($0000-$FFFF beim C64), ist diese Adresse zwei Byte lang (siehe oben). Der Programmzähler besteht aus zwei Registern, speziellen Speicherzellen, die wie üblich jeweils ein Byte aufnehmen können und die nieder- bzw. die höherwertige Hälfte der jeweiligen Adresse enthalten.

Nehmen wir an, der Programmzähler weist auf die Adresse $C000, an der sich der erste Befehl unseres Programms befindet (LDX #$00). Über den sogenannten Datenbus wird der Inhalt der Speicherzelle $C000 geholt, die Zahl $A2.

Die Zahl wird nun decodiert und als Befehl LDX erkannt. Anschließend wird der Programmzähler um eins erhöht und weist danach auf die Adresse $C001. Das Argument des Befehls LDX, die Zahl $00, befindet sich an dieser Adresse und wird ebenfalls mit Hilfe des Datenbusses vom Rechnerspeicher in die CPU übertragen.

Ob ein Befehls-Argument ein oder aber zwei Byte umfaßt, erkennt die CPU am Befehlscode. Nachdem das Befehlsargument vom Speicher zur CPU transportiert wurde, wird der Befehl ausgeführt.

Der Programmzähler wird bei jeder Übertragung eines Bytes um den Wert eins erhöht und weist somit nach der Bearbeitung eines Befehls (und seines Argumentes) auf den folgenden Befehl, der auf die gleiche Weise abgearbeitet wird.

Die CPU-Register

Im vorigen Abschnitt lernten wir zwei CPU-Register kennen, die zusammen den Programmzähler bilden. Die verschiedenen CPU-Register sind die wichtigsten Speicherzellen, die unser Rechner besitzt.

1. Der Akkumulator

Das wichtigste CPU-Register ist der sogenannte Akkumulator. Fast alle Operationen sind nur mit Hilfe des Akkumulators mög-

lich. Um beispielsweise eine bestimmte Speicherzelle mit einem beliebigen Wert zu beschreiben (analog dem POKE-Befehl), muß dieser Wert zuerst in den Akkumulator geladen werden. Anschließend kann der Akkumulatorinhalt in die gewünschte Speicherzelle übertragen werden. Außer für den Datentransfer ist der Akkumulator für Berechnungen aller Art zuständig. Additionen oder Subtraktionen können nur in diesem speziellen Register stattfinden.

2. Die Index-Register

Die CPU besitzt zwei Index-Register, die als X-Register und Y-Register bezeichnet werden. Beide Register werden vorwiegend als Schleifenzähler und zur Bearbeitung von Tabellen verwendet.

3. Das Status-Register

Das Status-Register enthält wie alle übrigen Register ein Byte. Dieses Byte gibt über den Prozessorstatus Auskunft, z.B. über das Ergebnis von Subtraktionen oder Additionen. Am Inhalt des Status-Registers können wir z.B. erkennen, ob das Ergebnis einer Addition größer oder kleiner als 255 ist oder ob eine Programmschleife beendet ist.

4. Der Stapelzeiger

Der Stapel ist ein spezieller, eine Seite (256 Byte) langer Speicherbereich, der zur kurzfristigen Ablage von Informationen geeignet ist und mit Hilfe des Stapelzeigers verwaltet wird.

4.4 Die Adressierungsarten

Von den 256 möglichen 8-Bit-Kombinationen stellen 151 einen gültigen Befehl für den 6510 dar. Dies beinhaltet mehrere gleichartige Befehle, die sich nur durch die Adressierungsart unterscheiden. Unterschiedliche Befehle gibt es 56 beim 6510. Dieser Befehlssatz ist leicht zu erlernen. Auf den folgenden Seiten finden Sie eine Übersicht der Befehle mit allen Adressierungsarten.

ADC - Add with carry

Addiert Operand + Carry-Flag zum Akku-Inhalt.

Adressierungsart	Schreibweise	Code
Unmittelbar	ADC #Op	$69
Absolut	ADC Op	$6D
Zeropage	ADC Op	$65
Abs.-X-Indiziert	ADC Op,X	$7D
Abs.-Y-Indiziert	ADC Op,Y	$79
Zerop.-X-Indiz.	ADC Op,X	$75
Indirekt-X-Ind.	ADC (Op,X)	$61
Indirekt-Y-Ind.	ADC (Op),Y	$71

AND - AND Accu

Verknüpft Akku-Inhalt mit Operand durch logisches UND.

Adressierungsart	Schreibweise	Code
Unmittelbar	AND #Op	$29
Absolut	AND Op	$2D
Zeropage	AND Op	$25
Abs.-X-Indiziert	AND Op,X	$3D
Abs.-Y-Indiziert	AND Op,Y	$39
Zerop.-X-Indiz.	AND Op,X	$35
Indirekt-X-Ind.	AND (Op,X)	$21
Indirekt-Y-Ind.	AND (Op),Y	$31

ASL - Arithmetic shift left

Verschiebt die Bits des Operanden um eine Stelle nach links.

Adressierungsart	Schreibweise	Code
Akku	ASL	$0A
Absolut	ASL Op	$0E
Zeropage	ASL Op	$06
Abs.-X-Indiziert	ASL Op,X	$1E
Zerop.-X-Indiz.	ASL Op,X	$16

BCC - Branch if carry clear

Verzweigt, falls das Carry-Flag gelöscht ist.

Adressierungsart	Schreibweise	Code
Relativ	BCC Op	$90

BCS - Branch if carry set

Verzweigt, falls das Carry-Flag gesetzt ist.

Adressierungsart	Schreibweise	Code
Relativ	BCS Op	$B0

BEQ - Branch if equal to zero

Verzweigt, falls das Zero-Flag gelöscht ist.

Adressierungsart	Schreibweise	Code
Relativ	BEQ Op	$F0

BIT - Test bits

Verknüpft Akku-Inhalt mit Operand durch logisches UND und setzt die entsprechenden Flags des Status-Registers.

Adressierungsart	Schreibweise	Code
Absolut	BIT Op	$2C
Zeropage	BIT Op	$24

BNE - Branch if not equal to zero

Verzweigt, falls das Zero-Flag gesetzt ist.

Adressierungsart	Schreibweise	Code
Relativ	BNE Op	$D0

BMI - Branch if minus

Verzweigt, falls das Negativ-Flag gesetzt ist.

Adressierungsart	Schreibweise	Code
Relativ	BMI Op	$30

BPL - Branch if plus

Verzweigt, falls das Negativ-Flag gelöscht ist.

Adressierungsart	Schreibweise	Code
Relativ	BPL Op	$10

BRK - Break

Programmunterbrechung.

Adressierungsart	Schreibweise	Code
Implizit	BRK	$00

BVS - Branch if overflow set

Verzweigt, falls das Overflow-Flag gesetzt ist.

Adressierungsart	Schreibweise	Code
Relativ	BVS Op	$70

CLC - Clear carry

Carry-Flag löschen.

Adressierungsart	Schreibweise	Code
Implizit	CLC	$18

CLD - Clear decimal mode

Dezimal-Flag löschen.

Adressierungsart	Schreibweise	Code
Implizit	CLD	$D8

CLI - Clear interrupt flag

Interrupt-Flag löschen.

Adressierungsart	Schreibweise	Code
Implizit	CLI	$58

CLV - Clear overflow flag

Overflow-Flag löschen.

Adressierungsart	Schreibweise	Code
Implizit	CLV	$B8

CMP - Compare with accu

Vergleicht Speicherzelle mit Akku.

Adressierungsart	Schreibweise	Code
Unmittelbar	CMP #Op	$C9
Absolut	CMP Op	$CD
Zeropage	CMP Op	$C5
Abs.-X-Indiziert	CMP Op,X	$DD
Abs.-Y-Indiziert	CMP Op,Y	$D9
Zerop.-X-Indiz.	CMP Op,X	$D5
Indirekt-X-Ind.	CMP (Op,X)	$C1
Indirekt-Y-Ind.	CMP (Op),Y	$D1

CPX - Compare with x-register

Vergleicht Speicherzelle mit X-Register.

Adressierungsart	Schreibweise	Code
Unmittelbar	CPX #Op	$E0
Absolut	CPX Op	$EC
Zeropage	CPX Op	$E4

CPY - Compare with y-register

Vergleicht Speicherzelle mit Y-Register.

Adressierungsart	Schreibweise	Code
Unmittelbar	CPY #Op	$C0
Absolut	CPY Op	$CC
Zeropage	CPY Op	$C4

DEC - Decrement memory

Dekrementiert den Inhalt der angegebenen Speicherzelle.

Adressierungsart	Schreibweise	Code
Absolut	DEC Op	$CE
Zeropage	DEC Op	$C6
Abs.-X-Indiziert	DEC Op,X	$DE
Zerop.-X-Indiz.	DEC Op,X	$D6

DEX - Decrement x-register

Dekrementiert den Inhalt des X-Registers.

Adressierungsart	Schreibweise	Code
Implizit	DEX	$CA

DEY - Decrement y-register

Dekrementiert den Inhalt des Y-Registers.

Adressierungsart	Schreibweise	Code
Implizit	DEY	$88

EOR - exclusive-or accu

Verknüpft Akku-Inhalt mit Operand durch logisches EXCLU-SIV-ODER.

Adressierungsart	Schreibweise	Code
Unmittelbar	EOR #Op	$49
Absolut	EOR Op	$4D
Zeropage	EOR Op	$45
Abs.-X-Indiziert	EOR Op,X	$5D
Abs.-Y-Indiziert	EOR Op,Y	$59
Zerop.-X-Indiz.	EOR Op,X	$55
Indirekt-X-Ind.	EOR (Op,X)	$41
Indirekt-Y-Ind.	EOR (Op),Y	$51

INC - Increment memory

Inkrementiert den Inhalt der angegebenen Speicherzelle.

Adressierungsart	Schreibweise	Code
Absolut	INC Op	$EE
Zeropage	INC Op	$E6
Abs.-X-Indiziert	INC Op,X	$FE
Zerop.-X-Indiz.	INC Op,X	$F6

INX - Increment x-register

Inkrementiert den Inhalt des X-Registers.

Adressierungsart	Schreibweise	Code
Implizit	INX	$E8

INY - Increment y-register

Inkrementiert den Inhalt des Y-Registers.

Adressierungsart	Schreibweise	Code
Implizit	INY	$C8

JMP - Jump

Verzweigt zur angegebenen Adresse.

Adressierungsart	Schreibweise	Code
Indirekt	JMP (Op)	$6C
Absolut	JMP Op	$4C

JSR - Jump to subroutine

Verzweigt zur angegebenen Adresse (Unterprogrammaufruf).

Adressierungsart	Schreibweise	Code
Absolut	JSR Op	$20

LDA - Load accu

Lädt Akku mit angegebenem Wert.

Adressierungsart	Schreibweise	Code
Unmittelbar	LDA #Op	$A9
Absolut	LDA Op	$AD
Zeropage	LDA Op	$A5
Abs.-X-Indiziert	LDA Op,X	$BD
Abs.-Y-Indiziert	LDA Op,Y	$B9
Zerop.-X-Indiz.	LDA Op,X	$B5
Indirekt-X-Ind.	LDA (Op,X)	$A1
Indirekt-Y-Ind.	LDA (Op),Y	$B1

LDX - Load x-register

Lädt X-Register mit angegebenem Wert.

Adressierungsart	Schreibweise	Code
Unmittelbar	LDX #Op	$A2
Absolut	LDX Op	$AE
Zeropage	LDX Op	$A6
Abs.-Y-Indiziert	LDX Op,Y	$BE
Zerop.-Y-Indiz.	LDX Op,Y	$B6

LDY - Load y-register

Lädt Y-Register mit angegebenem Wert.

Adressierungsart	Schreibweise	Code
Unmittelbar	LDY #Op	$A0
Absolut	LDY Op	$AC
Zeropage	LDY Op	$A4
Abs.-X-Indiziert	LDY Op,X	$BC
Zerop.-X-Indiz.	LDY Op,X	$B4

LSR - Logical shift right

Verschiebt die Bits des Operanden um eine Stelle nach rechts.

Adressierungsart	Schreibweise	Code
Akku	LSR	$4A
Absolut	LSR Op	$4E
Zeropage	LSR Op	$46
Abs.-X-Indiziert	LSR Op,X	$5E
Zerop.-X-Indiz.	LSR Op,X	$56

NOP - No operation

Keine Operation.

Adressierungsart	Schreibweise	Code
Implizit	NOP	$EA

ORA - OR accu

Verknüpft Akku-Inhalt mit Operand durch logisches ODER.

Adressierungsart	Schreibweise	Code
Unmittelbar	ORA #Op	$09
Absolut	ORA Op	$0D
Zeropage	ORA Op	$05
Abs.-X-Indiziert	ORA Op,X	$1D
Abs.-Y-Indiziert	ORA Op,Y	$19
Zerop.-X-Indiz.	ORA Op,X	$15
Indirekt-X-Ind.	ORA (Op,X)	$01
Indirekt-Y-Ind.	ORA (Op),Y	$11

PHA - Push accu

Bringt Akku-Inhalt auf Stack.

Adressierungsart	Schreibweise	Code
Implizit	PHA	$48

PHP - Push processor-status

Bringt Status-Register auf Stack.

Adressierungsart	Schreibweise	Code
Implizit	PHP	$08

PLA - Pull accu

Lädt Akku mit oberstem Stackelement.

Adressierungsart	Schreibweise	Code
Implizit	PLA	$68

PLP - Pull processor-status

Lädt Status-Register mit oberstem Stackelement.

Adressierungsart	Schreibweise	Code
Implizit	PLP	$28

ROL - Rotate left

Rotiert die Bits des Operanden um eine Stelle nach links.

Adressierungsart	Schreibweise	Code
Akku	ROL	$2A
Absolut	ROL Op	$2E
Zeropage	ROL Op	$26
Abs.-X-Indiziert	ROL Op,X	$3E
Zerop.-X-Indiz.	ROL Op,X	$36

ROR - Rotate right

Rotiert die Bits des Operanden um eine Stelle nach rechts.

Adressierungsart	Schreibweise	Code
Akku	ROR	$6A
Absolut	ROR Op	$6E
Zeropage	ROR Op	$66
Abs.-X-Indiziert	ROR Op,X	$7E
Zerop.-X-Indiz.	ROR Op,X	$76

RTI - Return from interrupt

Rücksprung aus Interrupt.

Adressierungsart	Schreibweise	Code
Implizit	RTI	$40

RTS - Return from subroutine

Rücksprung aus Unterprogramm.

Adressierungsart	Schreibweise	Code
Implizit	RTS	$60

SBC - Substract with carry

Subtrahiert Operand + Carry-Flag vom Akku-Inhalt.

Adressierungsart	Schreibweise	Code
Unmittelbar	SBC #Op	$E9
Absolut	SBC Op	$ED
Zeropage	SBC Op	$E5
Abs.-X-Indiziert	SBC Op,X	$FD
Abs.-Y-Indiziert	SBC Op,Y	$F9
Zerop.-X-Indiz.	SBC Op,X	$F5
Indirekt-X-Ind.	SBC (Op,X)	$E1
Indirekt-Y-Ind.	SBC (Op),Y	$F1

SEC - Set carry

Carry-Flag setzen.

Adressierungsart	Schreibweise	Code
Implizit	SEC	$38

SED - Set decimal mode

Dezimal-Flag setzen.

Adressierungsart	Schreibweise	Code
Implizit	SED	$F8

SEI - Set interrupt

Interrupt-Flag setzen.

Adressierungsart	Schreibweise	Code
Implizit	SEI	$78

STA - Store accu

Schreibt Akku-Inhalt in angegebene Speicherzelle.

Adressierungsart	Schreibweise	Code
Absolut	STA Op	$8D
Zeropage	STA Op	$85
Abs.-X-Indiziert	STA Op,X	$9D
Abs.-Y-Indiziert	STA Op,Y	$99
Zerop.-X-Indiz.	STA Op,X	$95
Indirekt-X-Ind.	STA (Op,X)	$81
Indirekt-Y-Ind.	STA (Op),Y	$91

STX - Store x-register

Schreibt X-Register in angegebene Speicherzelle.

Adressierungsart	Schreibweise	Code
Absolut	STX Op	$8E
Zeropage	STX Op	$86
Zerop.-Y-Indiz.	STX Op,Y	$96

STY - Store y-register

Schreibt Y-Register in angegebene Speicherzelle.

Adressierungsart	Schreibweise	Code
Absolut	STY Op	$8C
Zeropage	STY Op	$84
Zerop.-X-Indiz.	STY Op,X	$94

TAX - Transfer accu to x-register

Schreibt Akku-Inhalt ins X-Register.

Adressierungsart	Schreibweise	Code
Implizit	TAX	$AA

TAY - Transfer accu to y-register

Schreibt Akku-Inhalt ins Y-Register.

Adressierungsart	Schreibweise	Code
Implizit	TAY	$A8

TXA - Transfer x-register to accu

Schreibt X-Register-Inhalt in den Akku.

Adressierungsart	Schreibweise	Code
Implizit	TXA	$8A

TYA - Transfer y-register to accu

Schreibt Y-Register-Inhalt in den Akku.

Adressierungsart	Schreibweise	Code
Implizit	TYA	$98

TSX - Transfer stackregister to x-register

Schreibt Stackpointer-Inhalt ins X-Register.

Adressierungsart	Schreibweise	Code
Implizit	TSX	$BA

TXS - Transfer x-register to stackregister

Schreibt X-Register-Inhalt in den Stackpointer.

Adressierungsart	Schreibweise	Code
Implizit	TXS	$9A

4.5 Maschinenprogramme auf dem C64

Für den Fall, daß Sie noch nie (auch nicht in Form von Data-Zeilen) Kontakt mit einem Maschinenprogramm hatten, sollten Sie das weiter unten abgebildete Demo-Programm eingeben.

Das eigentliche Maschinenprogramm ist in der Data-Zeile am Ende des BASIC-Programms enthalten. Seine recht einfache Aufgabe besteht darin, die oberen Bildschirmzeilen mit dem Buchstaben A zu füllen. Sie können dieses Programm sofort eingeben und wie gewohnt mit RUN starten.

```
100 REM *** 'A' AUSGEBEN ***
110 FOR I=49152 TO 49152+10
120 : READ A:POKE I,A
130 NEXT
140 PRINT CHR$(147):REM SCREEN LOESCHEN
150 SYS 49152:REM PROGRAMM STARTEN
160 END
170 :
180 DATA 169,1,162,0,157,0,4,202,208,250,96
```

Zurück zu unserem Demo-Programm. BASIC-Programmierersind sicherlich über die enorme Geschwindigkeit erstaunt, mit der die 256 Zeichen ausgegeben werden. Irgendeine Verzögerung ist nicht festzustellen, das Programm arbeitet für das menschliche Auge praktisch in Nullzeit. Zum Vergleich ein BASIC-Programm, das analog dem Maschinenprogramm arbeitet:

```
100 REM *** 'A' AUSGEBEN IN BASIC ***
110 PRINT CHR$(147);:REM SCREEN LOESCHEN
120 FOR I=1 TO 256
130 : PRINT"A";
140 NEXT
```

Zugegeben: Auch das BASIC-Programm ist nicht allzu langsam (circa eine Sekunde zur Ausgabe aller A). Der Unterschied zwischen diesem und dem Maschinenprogramm ist dennoch erstaunlich.

4.6 Benutzung von Betriebssystem-Routinen

Der folgende Abschnitt führt alle besprochenen Betriebssystem-Routinen auf.

Routine: BSOUT ($FFD2)

Ausgabe eines Zeichens auf das Standardgerät "Bildschirm" oder eine geöffnete logische Datei, auf die zuvor mit CHKOUT die Ausgabe umgelenkt wurde.

```
Parameter hin:          ASCII-Code des Zeichens im Akkumulator.
Parameter zurück:       Keine
Beeinflußte Register:   Keine
```

Routine: BASIN ($FFCF)

Eingabe eines Zeichens vom Standardgerät "Tastatur" oder einer geöffneten logischen Datei, auf die zuvor mit CHKIN die Eingabe umgelenkt wurde.

```
Parameter hin:          Keine
Parameter zurück:       Zeichen im Akkumulator (ASCII-Code).
Beeinflußte Register:   Akkumulator, X-Register.
```

Routine: GETIN ($FFE4)

Zeichen von der Tastatur lesen.

```
Parameter hin:          Keine
Parameter zurück:       Zeichen im Akkumulator (ASCII-Code).
                        $00, wenn keine Taste gedrückt.
Beeinflußte Register:   Akkumulator, X-Register, Y-Register.
```

Routine: CHKOUT ($FFC9)

Leitet die Ausgabe auf eine zuvor geöffnete logische Datei um.

```
Parameter hin:          File-Nummer (Dateinummer) im X-Register.
Parameter zurück:       Keine
Beeinflußte Register:   Akkumulator, X-Register.
```

Routine: CHKIN ($FFC6)

Leitet die Eingabe auf eine zuvor geöffnete logische Datei um.

```
Parameter hin:          File-Nummer (Dateinummer) im X-Register.
Parameter zurück:       Keine
Beeinflußte Register:   Akkumulator, X-Register
```

Routine: CLRCH ($FFCC)

Schließt alle geöffneten Datenübertragungskanäle (ersetzt jedoch nicht das folgende Schließen der logischen Datei mit CLOSE) und lenkt die Ein-/Ausgabe auf die Standardgeräte Bildschirm und Tastatur um.

```
Parameter hin:          Keine
Parameter zurück:       Keine
Beeinflußte Register:   Akkumulator, X-Register.
```

Routine: PLOT ($FFF0)

Setzt den Cursor auf die angegebene Position oder holt die aktuelle Cursor-Position.

```
Parameter hin:          Cursor setzen: Carry-Flag löschen,
                        Spalte im Y-Register (0-39), Zeile im X-
                        Register (0-24). Cursor-Position holen:
                        Carry-Flag setzen.
Parameter zurück:       Cursor-Position holen: Spalte im Y-Register,
                        Zeile im X-Register.
Beeinflußte Register:   Akkumulator, X-Register, Y-Register.
```

Die Routinen des BASIC-Interpreters

Die Routinen des BASIC-Interpreters unterscheiden sich leider von Rechner zu Rechner. Die Funktionsweise ist jedoch immer identisch, ebenso die hin- bzw. zurückübergebenen Parameter.

Sie dürfen davon ausgehen, daß jede der im folgenden genannten Routinen alle Register beeinflußt!

Routine: CHKKOM
Adresse: $AEFD

Liest das aktuelle Zeichen aus dem BASIC-Text und prüft, ob es sich um ein Komma handelt.

```
Parameter hin:          Keine
Parameter zurück:       Keine
```

Routine: GETBYT
Adresse: $B79E

Liest einen Ein-Byte-Wert aus dem BASIC-Text oder eine numerische Variable mit entsprechendem Inhalt (0-255).

```
Parameter hin:          Keine
Parameter zurück:       Byte-Wert im X-Register.
```

Routine: FRMNUM
Adresse: $AD8A

Wertet einen beliebigen numerischen Ausdruck aus, der sich im BASIC-Text befindet.

```
Parameter hin:          Keine
Parameter zurück:       Ergebnis im Fließkommaformat im FAC
                        (Fließkomma-Akkumulator).
```

Routine: ADRFOR
Adresse: $B7F7

Wandelt eine Fließkommazahl, die sich im FAC befindet, ins Adreßformat (Low-Byte/High-Byte) und kann sehr effektiv zusammen mit FRMNUM zum Einlesen eines Zwei-Byte-Wertes aus dem BASIC-Text eingesetzt werden.

```
Parameter hin:          Fließkommazahl im FAC.
Parameter zurück:       Integer-Zahl im Adreßformat im Y-Register
                        (Low-Byte) und im Akkumulator (High-Byte).
```

Routine: ADRBYT
Adresse: $B7EB

Liest einen Zwei-Byte-Wert und einen folgenden - durch ein Komma getrennten - Ein-Byte-Wert aus dem BASIC-Text ein.

```
Parameter hin:          Keine
Parameter zurück:       Zwei-Byte-Wert im Adreßformat in $14/$15
                        und Ein-Byte-Wert im X-Register.
```

Routine: GETPOS
Adresse: $B08B

Liest eine Variable beliebigen Typs aus dem BASIC-Text und übergibt einen Zeiger auf die Variable (numerische Variablen) bzw. auf die Descriptoren (String-Variablen).

```
Parameter hin:          Keine
Parameter zurück:       Zeiger in Akkumulator (Low-Byte) und im Y-
                        Register (High-Byte).
```

Routine: STRRES
Adresse: $B4F4

Reserviert Platz für einen anzulegenden String und vermindert zugleich den Zeiger auf das Ende des String-Stacks um die String-Länge. Ist nicht ausreichend Platz vorhanden, gibt STRRES die Fehlermeldung OUT OF MEMORY ERROR aus.

```
Parameter hin:          String-Länge im Akkumulator
Parameter zurück:       Keine
```

Routine: INTOUT
Adresse: $BDCD

Gibt eine Integer-Zahl ab der aktuellen Cursor-Position auf dem Bildschirm aus.

```
Parameter hin:          Integer-Zahl im X-Register (Low-Byte) und
                        Akkumulator (High-Byte).
Parameter zurück:       Keine
```

4.7 Weitere Beispiele und BASIC-Ladeprogramme

Mehrmals in diesem Buch versprach ich Ihnen, auf ein Thema einzugehen, das leider in kaum einem Assembler-Lehrbuch beschrieben wird, auf die Zusammenarbeit von Assembler-Routinen und BASIC-Programmen.

Im folgenden Teil wird Ihnen das nötige Handwerkszeug vermittelt, um z.B. Eingabe- oder Sortierroutinen in Assembler zu schreiben, die von einem BASIC-Programm genutzt werden.

Die in diesem dritten Teil entwickelten Programme sind deutlich komplexer als die vorangegangenen und werden deshalb in verschiedenen Teilschritten entwickelt und erläutert. Am Ende eines jeden Abschnitts wird das Listing des kompletten Programms vorgestellt.

Vor der Entwicklung diverser Programme ist jedoch ein wenig Theorie unumgänglich, da Sie unbedingt darüber Bescheid wissen müssen, wie der BASIC-Interpreter einen BASIC-Text liest und Variablen organisiert.

Da man ja bekanntlich durch die Praxis immer noch am besten lernt, möchte ich Ihnen nun die ersten fünf einer Vielzahl von äußerst nützlichen und "gebrauchsfertigen" Assembler-Routinen vorstellen. Gebrauchsfertig soll heißen, daß Sie diese Programme ohne jede Änderungen oder Ergänzungen sofort nutzen können. Alle Routinen bilden jeweils eine geschlossene Einheit und greifen nicht auf andere Routinen zu (mit Ausnahme natürlich der Betriebssystem-Routinen, die ja ohnehin vorhanden sind).

Alle Routinen dienen demselben Zweck: Sie machen Ihnen die verborgenen, vom BASIC 2.0 nicht unterstützten Fähigkeiten des Commodore 64 verfügbar. Damit auch die Nicht- oder Noch-Nicht-Assemblerprogrammierer von diesen Programmen etwas haben, finden Sie im Anschluß an jedes Assemblerlisting einen sogenannten BASIC-Lader zu dem Programm.

Was ist nun ein BASIC-Lader? In jedem BASIC-Lader ist der Programmcode des Maschinenprogramms in Form von DATA-

Zeilen abgelegt. Nachdem Sie den BASIC-Lader abgetippt, auf Diskette gespeichert (nicht vergessen!) und gestartet haben, werden diese DATAs in einer Schleife mittels READ gelesen und in die festgelegten Speicherzellen gepoket. Anschließend können Sie den BASIC-Lader löschen. Die Assembler-Routine kann nun mit dem passenden SYS-Befehl, den Sie im Kopf des BASIC-Laders finden, aufgerufen werden.

Diese ganze Prozedur (BASIC-Lader abtippen bzw. laden, starten und löschen) müssen Sie aber nur einmal durchführen. Etwas weiter unten werde ich Ihnen zeigen, wie Sie die Programme, nachdem sie im Speicher abgelegt wurden, direkt als Maschinenprogramm auf Diskette speichern und später auch wieder laden können.

Wenn Sie am Ende des Buches alle Programme abgetippt haben, können Sie sich die einzelnen Routinen auch als ein Gesamtprogramm abspeichern. Alle Routinen sind nämlich so codiert, daß sie "nahtlos" aneinanderpassen und alle zusammen verwendet werden können. Als Speicher habe ich den freien RAM-Speicher ab Speicherzelle 49.152 genommen. Dieser vier KByte große Speicherbereich wird von BASIC nicht verwendet und eignet sich daher sehr gut zur Ablage von Maschinenroutinen.

Der angegebene Speicherbereich gilt natürlich nur für die BASIC-Lader. Wenn Sie die Assemblerlistings abtippen, können Sie sich die Programme in jeden beliebigen Speicherbereich "assemblieren" lassen. Dazu müssen Sie nur die Startadresse am Anfang des Listings entsprechend ändern.

Um Komplikationen bei der Eingabe der Assemblerprogramme zu vermeiden, habe ich die Listings, was die Verwendung von Labels und Steueranweisungen betrifft, bewußt sehr "schlicht" gehalten. Auch wenn Ihnen nur ein sogenanntes Monitorprogramm (darunter versteht man eine Kombination aus einem meist sehr einfach gebauten Assembler und einem Diassembler) zur Verfügung steht, sollten Sie die Programme problemlos verarbeiten können.

Noch ein Wort zu den BASIC-Ladern: Gerade beim Eintippen langer Zahlenkolonnen, wie sie in den DATA-Zeilen stehen, vertippt man sich nur zu leicht. Aus diesem Grund steht am Ende jeder Zeile eine sog. "Prüfsumme, die die Summe der in dieser Zeile stehenden DATA-Werte darstellt. Beim Einlesen der DATAs wird nun kontrolliert, ob die aufaddierten Werte mit der Prüfsumme übereinstimmen. Falls nicht, bricht das Programm mit einer entsprechenden Fehlermeldung ab. Da die Fehlermeldung die Nummer der fehlerhaften Zeile enthält, müssen Sie jeweils immer nur zehn DATA-Werte (soviele stehen in einer Programmzeile) auf Tippfehler überprüfen. Anschließend starten Sie das Programm neu. Doch genug der Vorrede, schauen wir uns die ersten beiden Assembler-Routinen an.

```
100 ;**********************************************************
105 ;*                                                        *
110 ;* programm: dsavem                                       *
120 ;* speichert einen beliebigen speicherbereich             *
130 ;* auf disk ab                                            *
140 ;*                                                        *
150 ;**********************************************************
160 ;*                                                        *
170 ;* aufruf: sys 49489,pr$,ba,be                            *
175 ;* pr$: name des programms                                *
180 ;* ba: blockanfang                                        *
185 ;* be: blockende                                          *
190 ;*                                                        *
195 ;**********************************************************
200 ;
205 ;
210 .ba 49489    ;*** startadresse ***
220 ;
230 ;
240 ;*** labels ***********************************************
250 .gl zp1 = 251   ;zwischenspeicher
260 .gl zp2 = 252
270 ;
280 ;
290 ;*** parameter einlesen ***********************************
300           jsr $aefd       ;auf komma testen
310           jsr $ad9e       ;file-namen
320           jsr $b6a3       ;holen
330           jsr $ffbd       ;und setzen
340           ldx #8          ;floppy-geraeteadresse
350           jsr $ffba       ;setzen
360           jsr $aefd       ;auf komma testen
370           jsr $ad8a       ;blockanfang holen
380           jsr $b7f7       ;nach integer wandeln
```

```
390          lda $14      ;wert umspeichern
400          sta zp1
410          lda $15
420          sta zp2
430          jsr $aefd    ;auf komma testen
440          jsr $ad8a    ;blockende holen
450          jsr $b7f7    ;nach integer wandeln
460          sec          ;blockende>=blockanfang?
470          lda $15      ;high-Bytes
480          sbc zp2
490          beq ds1
500          bcs ds2
510          jmp $b248    ;nein, dann 'illegal quantity'
520 ds1      sec
530          lda $14      ;low-Bytes
540          sbc zp1
550          bcs ds2
560          jmp $b248    ;nein, dann 'illegal quantity'
570 ds2      ldx $14      ;blockende um 1 erhoehen
580          ldy $15
590          inx
600          bne ds3
610          iny
620 ds3      lda #zp1     ;zeiger auf blockanfang
630 ;
640 ;*** save-routine ****************************************
650          stx $ae      ;blockende speichern
660          sty $af
670          tax          ;zeiger auf blockanfang
680          lda $00,x    ;blockanfang umspeichern
690          sta $c1
700          lda $01,x
710          sta $c2
720          lda #$61     ;secundaeradresse 1
730          sta $b9
740          ldy $b7      ;laenge des file-namens
750          bne ds4
760          jmp $f710    ;'missing filename error'
770 ds4      jsr $f3d5    ;file-namen senden
780          jsr $f68f    ;'saving' ausgeben
790          lda $ba      ;geraeteadresse
800          jsr $ed0c    ;listen senden
810          lda $b9      ;secundaeradresse
820          jsr $edb9    ;fuer listen senden
830          ldy #0
840          jsr $fb8e    ;blockanfang nach $ac/ad
850          lda $ac      ;blockanfang (low)
860          jsr $eddd    ;senden
870          lda $ad      ;(high)
880          jsr $eddd    ;senden
890 ds5      jsr ds8      ;blockende erreicht?
900          bcs ds7      ;ja, dann fertig
910          sei          ;auf ram umschalten
```

```
920         lda #53
930         sta $01
940         lda ($ac),y     ;programm-Byte
950         pha             ;zwischenspeichern
960         lda #55         ;auf rom zurueckschalten
970         sta $01
980         cli
990         pla             ;programm-Byte zurueckholen
1000        jsr $eddd       ;und senden
1010        jsr $ffe1       ;stop-taste pruefen
1020        bne ds6         ;nicht gedrueckt, dann weiter
1030        jmp $f633       ;zur original-routine
1040 ds6    jsr $fcdb       ;aktuelle adresse erhoehen
1050        bne ds5         ;<>0, dann weiter
1060 ds7    jsr $f63f       ;zur original-routine
1070        bcc dsfr        ;kein fehler
1080        jmp $e104       ;zur fehlerauswertung
1090 dsfr   rts             ;fertig!
1100 ;* endadresse erreicht? *
1110 ds8    lda $af         ;endadresse (high)
1120        bne ds9
1130        lda $ae         ;low
1140        bne ds9
1150        clc
1160        rts             ;endadresse=0, dann fertig
1170 ds9    jmp $fcd1       ;zur original-routine
```

Und das ganze als BASIC-Lader:

```
100 rem ****************************************************
105 rem *                                                  *
110 rem * programm: dsavem                                 *
120 rem * programmlaenge: 187 bytes                        *
130 rem * speichert einen beliebigen speicherbereich       *
140 rem * auf disk ab                                      *
150 rem *                                                  *
200 rem ****************************************************
205 rem *                                                  *
210 rem * aufruf: sys 49489,pr$,ba,be                      *
220 rem * pr$: name des programms                          *
230 rem * ba: blockanfang                                  *
240 rem * be: blockende                                    *
265 rem *                                                  *
270 rem ****************************************************
280 :
290 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=49489:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
```

```
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 32,253,174,32,158,173,32,163,182,32,1231
1010 data 189,255,162,8,32,186,255,32,253,174,1546
1020 data 32,138,173,32,247,183,165,20,133,251,1374
1030 data 165,21,133,252,32,253,174,32,138,173,1373
1040 data 32,247,183,56,165,21,229,252,240,5,1430
1050 data 176,13,76,72,178,56,165,20,229,251,1236
1060 data 176,3,76,72,178,166,20,164,21,232,1108
1070 data 208,1,200,169,251,134,174,132,175,170,1614
1080 data 181,0,133,193,181,1,133,194,169,97,1282
1090 data 133,185,164,183,208,3,76,16,247,32,1247
1100 data 213,243,32,143,246,165,186,32,12,237,1509
1110 data 165,185,32,185,237,160,0,32,142,251,1389
1120 data 165,172,32,221,237,165,173,32,221,237,1655
1130 data 32,255,193,176,30,120,169,53,133,1,1162
1140 data 177,172,72,169,55,133,1,88,104,32,1003
1150 data 221,237,32,225,255,208,3,76,51,246,1554
1160 data 32,219,252,208,221,32,63,246,144,3,1420
1170 data 76,4,225,96,165,175,208,6,165,174,1294
1180 data 208,2,24,96,76,209,252,0,0,0,867
2000 data -1:rem endmarkierung
100 ;*********************************************************
105 ;*                                                       *
110 ;* programm: dloadm                                      *
120 ;* laedt einen beliebigen speicherbereich                *
130 ;* von disk in den rechner                               *
140 ;*                                                       *
150 ;*********************************************************
160 ;*                                                       *
170 ;* aufruf: sys 49676,pr$,ba                              *
175 ;* pr$: name des programms                               *
180 ;* ba: blockanfang                                       *
190 ;*                                                       *
195 ;*********************************************************
200 ;
205 ;
210 .ba 49676    ;*** startadresse ***
220 ;
230 ;
240 ;*** labels *********************************************
250 .gl zp1 = 251    ;zwischenspeicher
260 .gl zp2 = 252
270 ;
280 ;
290 ;*** parameter einlesen **********************************
300          lda #0         ;flag fuer
```

```
310             sta $0a         ;load
320             jsr $aefd       ;auf komma testen
330             jsr $ad9e       ;file-namen
340             jsr $b6a3       ;holen
350             jsr $ffbd       ;und setzen
360             ldx #8          ;floppy-geraeteadresse
370             ldy #0          ;secundaeradresse
380             jsr $ffba       ;setzen
390             jsr $aefd       ;auf komma testen
400             jsr $ad8a       ;blockanfang holen
410             jsr $b7f7       ;nach integer wandeln
420             ldx $14         ;blockanfang
430             ldy $15
440             lda $0a
445 ;*** load-routine ***************************************
450             jsr $ffd5       ;load-routine aufrufen
460             bcc dl1         ;kein fehler
470             jmp $e104       ;zur fehlerauswertung
480 dl1         lda $0a         ;load-flag
490             jsr $ffb7       ;status holen
500             and #$bf        ;eof-bit löeschen
510             beq dl2         ;kein fehler
520             ldx #$1d        ;nummer fuer 'load error'
530             jmp $a437       ;fehlermeldung ausgeben
540 dl2         rts             ;fertig!
```

Und der BASIC-Lader:

```
100 rem ****************************************************
105 rem *                                                  *
110 rem * programm: dloadm                                 *
120 rem * programmlaenge: 61 bytes                         *
130 rem * laedt einen beliebigen speicherbereich           *
140 rem * von disk in den rechner                          *
150 rem *                                                  *
200 rem ****************************************************
205 rem *                                                  *
210 rem * aufruf: sys 49676,pr$,ba,be                      *
220 rem * pr$: name des programms                          *
230 rem * ba: blockanfang                                  *
265 rem *                                                  *
270 rem ****************************************************
280 :
290 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=49676:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
```

```
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 169,0,133,10,32,253,174,32,158,173,1134
1010 data 32,163,182,32,189,255,162,8,160,0,1183
1020 data 32,186,255,32,253,174,32,138,173,32,1307
1030 data 247,183,166,20,164,21,165,10,32,213,1221
1040 data 255,144,3,76,4,225,165,10,32,183,1097
1050 data 255,41,191,240,5,162,29,76,55,164,1218
1060 data 96,0,0,0,0,0,0,0,0,0,96
2000 data -1:rem endmarkierung
```

Diese beiden Routinen erlauben es Ihnen, beliebige Speicherbereiche auf Diskette abzuspeichern bzw. zu laden. Das Besondere bei DSAVEM ist dabei, daß sich der gesamte RAM-Speicher ansprechen läßt, also auch die RAM-Bereiche "unter" dem BASIC- und Kernel-ROM! Wenn Sie also zum Beispiel unter dem BASIC-ROM eine hochauflösende Grafik abgelegt haben, so können Sie diese mit DSAVEM problemlos auf Diskette sichern. Auch DLOADM weist eine Besonderheit auf: Im Gegensatz zum bekannten "LOAD "NAME",8,1" werden bei DLOADM die BASIC-Zeiger nicht verändert. Der gefürchtete OUT OF MEMORY ERROR nach dem Einladen eines Maschinenprogramms tritt daher bei DLOADM nicht auf.

Wenn Sie die BASIC-Lader der Programme abgetippt haben, dann habe ich gleich eine praktische Anwendung für DSAVEM. Wie ich schon erwähnt habe, ist es recht mühsam immer wieder den BASIC-Lader zu laden und das Maschinenprogramm in den Speicher poken zu lassen. Deshalb werde ich Ihnen jetzt zeigen, wie Sie die Programme direkt auf Diskette bekommen. Als Vorbereitung starten Sie bitte nun den BASIC-Lader von DSAVEM, damit sich das Programm im Speicher befindet. Wir werden es nämlich gleich benötigen. Danach gehen Sie wie folgt vor:

Im Kopfbereich jedes BASIC-Laders finden Sie unter anderem die Startadresse der Routine (hinter dem SYS-Befehl) sowie ihre Länge. DSAVEM beispielsweise beginnt bei Adresse 49.489 und hat eine Länge von 187 Bytes. Diese beiden Werte sind schon

ie brauchen. Bitte geben Sie einfach die folgende 'leich im Direktmodus) ein:

```
.ى),"DSAVEM",49489,49489+187-1
```

Das ist schon alles. Die Routine DSAVEM wird nun unter dem Namen "DSAVEM" auf Diskette gespeichert. Um das Programm später wieder einzuladen, verwenden Sie DLOADM:

```
SYS 49676,"DSAVEM",49489
```

Voraussetzung dafür ist natürlich, daß Sie zuvor DLOADM in den Speicher geholt haben! Dazu müssen Sie notgedrungen auf das gewohnte "LOAD "DLOADM",8,1" zurückgreifen und anschließend mit NEW die BASIC-Zeiger zurücksetzen.

Das ist ein Fehler, der einem leicht passiert. Man ruft von einem BASIC-Programm aus eine Maschinensprache-Routine auf und hat vergessen, diese zuvor in den Speicher zu laden. In den meisten Fällen hat dies einen Systemabsturz zur Folge, da in den entsprechenden Speicherzellen irgendwelche undefinierten Werte stehen. Um das zu vermeiden, sollte man alle benötigten Routinen immer ganz am Anfang eines BASIC-Programms einladen lassen.

Der eben vorgestellte Weg, ein durch einen BASIC-Lader in den Speicher gepoketes Maschinenprogramm auf Diskette abzuspeichern, gilt für alle in diesem Buch vorgestellten Routinen:

```
SYS 49489,"NAME",Startadresse,Startadresse+Länge-1
```

Für NAME, Startadresse und Länge setzen Sie jeweils die im Kopf des BASIC-Laders angegebenen Werte ein. Aber, wie gesagt, bitte vergessen Sie nicht, zuvor DSAVEM in den Rechner zu laden!

Die nächsten beiden Routinen sind nicht weniger interessant als DSAVEM und DLOADM.

```
100 ;**********************************************************
105 ;*                                                        *
110 ;* programm: transfer                                     *
120 ;* verschiebt beliebige speicherbereiche                  *
130 ;*                                                        *
140 ;**********************************************************
150 ;*                                                        *
160 ;* aufruf: sys 49152,ba,be,na,kn                          *
165 ;* ba: blockanfang (quelle)                               *
170 ;* be: blockende (quelle)                                 *
175 ;* na: blockanfang (ziel)                                 *
180 ;* kn: speicherart                                        *
185 ;*     (1=zeichengenerator, 3=ram, 5=rom)                 *
190 ;*                                                        *
195 ;**********************************************************
200 ;
205 ;
210 .ba 49152   ;*** startadresse ***
220 ;
230 ;
240 ;*** labels ***********************************************
245 .gl zp1 = 251   ;zwischenspeicher
250 .gl zp2 = 252
255 .gl zp3 = 253
260 .gl zp4 = 254
270 ;
280 ;
320 ;*** parameter einlesen ************************************
330           jsr $aefd      ;auf komma testen
340           jsr $ad8a      ;blockanfang holen
350           jsr $b7f7      ;nach integer wandeln
360           lda $14        ;wert umspeichern
370           sta zp1
380           lda $15
390           sta zp2
400           jsr $aefd      ;auf komma testen
410           jsr $ad8a      ;blockende holen
420           jsr $b7f7      ;nach integer wandeln
430           sec            ;blockende >= blockanfang?
440           lda $15        ;high-Bytes
450           sbc zp2
460           beq tr1
470           bcs tr2
480           jmp $b248      ;nein, dann 'illegal quantity'
490 tr1       sec
500           lda $14        ;low-Bytes
510           sbc zp1
520           bcs tr2
530           jmp $b248      ;nein, dann 'illegal quantity'
540 tr2       ldy $14        ;blockende um 1 erhoehen ...
550           ldx $15
560           iny            ;low-Byte
570           bne tr3        ;<>0, dann fertig
```

```
580            inx            ;high-Byte
590 tr3        sty zp3        ;... und zwischenspeichern
600            stx zp4
610            jsr $aefd      ;auf komma testen
620            jsr $ad8a      ;neuen blockanfang holen
630            jsr $b7f7      ;nach integer wandeln
640            pha            ;auf stack zwischenspeichern
650            tya
660            pha
670            jsr $aefd      ;auf komma testen
680            jsr $b79e      ;speicherart-Flag holen
690            cpx #1         ;=1?
700            beq tr4
710            cpx #3         ;=3?
720            beq tr4
730            cpx #5         ;=5?
740            beq tr4
750            jmp $b248      ;nein, dann 'illegal quantity'
760 ;
770 ;*** parameter setzen ************************************
780 tr4        pla            ;neuer blockanfang
790            sta $58
800            pla
810            sta $59
820            lda zp1        ;alter blockanfang
830            sta $5f
840            lda zp2
850            sta $60
860            lda zp3        ;altes blockende
870            sta $5a
880            lda zp4
890            sta $5b
900            txa            ;speicherart-Flag
910            clc
920            adc #50
930            sei            ;interrupt sperren
940            sta $01        ;speicherkonfiguration setzen
950 ;
960 ;*** transfer-routine ************************************
970            sec            ;blockanfang ermitteln
980            lda $5a
990            sbc $5f
1000           sta $22
1010           lda $5b
1020           sbc $60
1030           sta $23
1040           sec            ;anfangsadresse vergleichen
1050           lda $60        ;high-Bytes
1060           sbc $59
1070           beq tr5
1080           bcs tr6
1090           jmp tr11
1100 tr5       sec
```

```
1110          lda $5f        ;low-Bytes
1120          sbc $58
1130          beq tr11
1140          bcs tr6
1150          jmp tr11
1160 ;* neuer blockanfang <= altem, daher von der *
1170 ;* anfangsadresse an umspeichern *
1180 tr6      ldy #0
1190          ldx $23        ;anzahl der pages
1200          beq tr8        ;keine ganze page vorhanden
1210 tr7      lda ($5f),y    ;ein byte verschieben
1220          sta ($58),y
1230          iny
1240          bne tr7        ;naechstes byte
1250          inc $60
1260          inc $59
1270          dex            ;pagezaehler
1280          bne tr7        ;naechste page
1290 tr8      ldx $22        ;laenge der restpage
1300          beq tr10       ;=0, dann fertig
1310 tr9      lda ($5f),y    ;ein byte verschieben
1320          sta ($58),y
1330          iny
1340          dex            ;restzaehler
1350          bne tr9        ;naechstes byte
1360 tr10     lda #$37       ;alte speicherkonfiguration
1370          sta $01        ;wiederherstellen
1380          cli            ;interrupt freigeben
1390          rts            ;fertig!
1400 ;* neuer blockanfang > altem, daher von der *
1410 ;* endadresse an umspeichern *
1420 tr11     lda $23        ;zeiger auf letzte
1430          clc            ;quellpage richten
1440          adc $60
1450          sta $60
1460          lda $23
1470          clc
1480          adc $59
1490          sta $59
1500          ldy $22        ;restlaenge
1510          beq tr13       ;kein rest vorhanden
1520 tr12     dey
1530          lda ($5f),y    ;ein byte verschieben
1540          sta ($58),y
1550          cpy #0         ;rest umgespeichert?
1560          bne tr12       ;nein
1570 tr13     ldx $23        ;anzahl pages
1580          beq tr16       ;=0, dann fertig
1590 tr14     dec $60
1600          dec $59
1610 tr15     dey
1620          lda ($5f),y    ;ein byte verschieben
1630          sta ($58),y
```

```
1640          cpy #0          ;rest umgespeichert?
1650          bne tr15        ;nein
1660          dex
1670          bne tr14        ;naechste page
1680 tr16     lda #$37        ;alte speicherkonfiguration
1690          sta $01         ;wiederherstellen
1700          cli             ;interrupt freigeben
1710          rts             ;fertig!
```

Der BASIC-Lader:

```
100 rem ****************************************************
105 rem *                                                  *
110 rem * programm: transfer                               *
120 rem * programmlaenge: 250 bytes                        *
130 rem * verschiebt beliebige speicherbereiche            *
135 rem *                                                  *
200 rem ****************************************************
205 rem *                                                  *
210 rem * aufruf: sys 49152,ba,be,na,kn                    *
220 rem * ba: blockanfang (quelle)                         *
230 rem * be: blockende (quelle)                           *
240 rem * na: blockanfang (ziel)                           *
250 rem * kn: speicherart                                  *
260 rem *     (1=zeichengenerator, 3=ram, 5=rom)           *
265 rem *                                                  *
270 rem ****************************************************
280 :
290 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=49152:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 32,253,174,32,138,173,32,247,183,165,1429
1010 data 20,133,251,165,21,133,252,32,253,174,1434
1020 data 32,138,173,32,247,183,56,165,21,229,1276
1030 data 252,240,5,176,13,76,72,178,56,165,1233
1040 data 20,229,251,176,3,76,72,178,164,20,1189
1050 data 166,21,200,208,1,232,132,253,134,254,1601
1060 data 32,253,174,32,138,173,32,247,183,72,1336
1070 data 152,72,32,253,174,32,158,183,224,1,1281
1080 data 240,11,224,3,240,7,224,5,240,3,1197
```

```
1090 data 76,72,178,104,133,88,104,133,89,165,1142
1100 data 251,133,95,165,252,133,96,165,253,133,1676
1110 data 90,165,254,133,91,138,24,105,50,120,1170
1120 data 133,1,56,165,90,229,95,133,34,165,1101
1130 data 91,229,96,133,35,56,165,96,229,89,1219
1140 data 240,5,176,15,76,197,192,56,165,95,1217
1150 data 229,88,240,43,176,3,76,197,192,160,1404
1160 data 0,166,35,240,14,177,95,145,88,200,1160
1170 data 208,249,230,96,230,89,202,208,242,166,1920
1180 data 34,240,8,177,95,145,88,200,202,208,1397
1190 data 248,169,55,133,1,88,96,165,35,24,1014
1200 data 101,96,133,96,165,35,24,101,89,133,973
1210 data 89,164,34,240,9,136,177,95,145,88,1177
1220 data 192,0,208,247,166,35,240,16,198,96,1398
1230 data 198,89,136,177,95,145,88,192,0,208,1328
1240 data 247,202,208,240,169,55,133,1,88,96,1439
2000 data -1:rem endmarkierung
100 ;*********************************************************
105 ;*                                                       *
110 ;* programm: myfill                                      *
120 ;* fuellt beliebige speicherbereiche mit einem wert      *
130 ;*                                                       *
140 ;*********************************************************
150 ;*                                                       *
160 ;* aufruf: sys 49402,ba,be,wt                            *
170 ;* ba: blockanfang                                       *
175 ;* be: blockende                                         *
180 ;* wt: fuellwert (0-255)                                 *
190 ;*                                                       *
195 ;*********************************************************
200 ;
205 ;
210 .ba 49402   ;*** startadresse ***
220 ;
230 ;
240 ;*** labels *********************************************
250 .gl zp1 = 251   ;zwischenspeicher
260 .gl zp2 = 252
270 ;
280 ;
320 ;*** parameter einlesen *********************************
330          jsr $aefd       ;auf komma testen
340          jsr $ad8a       ;blockanfang holen
350          jsr $b7f7       ;nach integer wandeln
360          lda $14         ;wert umspeichern
370          sta zp1
380          lda $15
390          sta zp2
400          jsr $aefd       ;auf komma testen
410          jsr $ad8a       ;blockende holen
420          jsr $b7f7       ;nach integer wandeln
430          sec             ;blockende >= blockanfang
440          lda $15         ;high-Bytes
```

```
450            sbc zp2
460            beq mf1
470            bcs mf2
480            jmp $b248       ;nein, dann 'illegal quantity'
490 mf1        sec
500            lda $14         ;low-Bytes
510            sbc zp1
520            bcs mf2
530            jmp $b248       ;nein, dann 'illegal quantity'
540 mf2        jsr $aefd       ;auf komma testen
550            jsr $b79e       ;fuellwert holen
560            stx zw          ;zwischenspeichern
570 ;
580 ;*** fuell-routine *************************************
590            ldy #0
600 mf3        lda zw          ;fuellwert
610            sta (zp1),y     ;abspeichern
620            lda $15         ;endadresse erreicht?
630            cmp zp2
640            beq mf4
650            jmp mf5
660 mf4        lda $14
670            cmp zp1
680            bne mf5
690            rts             ;ja, dann fertig!
700 mf5        inc zp1
710            bne mf6
720            inc zp2
730 mf6        jmp mf3         ;nein, dann weiter
```

Der BASIC-Lader:

```
100 rem ****************************************************
105 rem *                                                  *
110 rem * programm: myfill                                 *
120 rem * programmlaenge: 87 bytes                         *
130 rem * fuellt beliebige speicherbereiche mit einem wert *
140 rem *                                                  *
200 rem ****************************************************
205 rem *                                                  *
210 rem * aufruf: sys 49402,ba,be,wt                       *
220 rem * ba: blockanfang                                  *
230 rem * be: blockende                                    *
240 rem * wt: fuellwert (0-255)                            *
250 rem *                                                  *
270 rem ****************************************************
280 :
290 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=49402:rem startadresse der routine
430 ps=0:z=0
```

```
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 32,253,174,32,138,173,32,247,183,165,1429
1010 data 20,133,251,165,21,133,252,32,253,174,1434
1020 data 32,138,173,32,247,183,56,165,21,229,1276
1030 data 252,240,5,176,13,76,72,178,56,165,1233
1040 data 20,229,251,176,3,76,72,178,32,253,1290
1050 data 174,32,158,183,134,2,160,0,165,2,1010
1060 data 145,251,165,21,197,252,240,3,76,72,1422
1070 data 193,165,20,197,251,208,1,96,230,251,1612
1080 data 208,2,230,252,76,52,193,0,0,0,1013
2000 data -1:rem endmarkierung
```

Sicher sind Ihnen beim Durchlesen oder Abtippen der Listings schon viele Anwendungsmöglichkeiten für diese beiden Routinen eingefallen. Mit MYFILL lassen sich beispielsweise blitzschnell der Grafikspeicher oder Teile des Bildschirms löschen. Mit TRANSFER können Sie beliebige Speicherbereiche verschieben, wobei sich der Quellbereich und der Zielbereich sogar auch überlappen dürfen. Probieren Sie doch einmal TRANSFER 1024,2023,1025,3. Dadurch wird der Inhalt des Textbildschirms um eine Stelle nach rechts verschoben. MYFILL und TRANSFER werden uns später bei der Grafikprogrammierung eine große Hilfe sein. Sie sollten die beiden Routinen daher unbedingt eintippen, bevor Sie Kapitel 6 durcharbeiten.

Haben Sie nicht manchmal auch schon gedacht, daß es doch ganz günstig wäre, wenn man die Funktionstasten des Commodore 64 mit häufig benötigten Befehlsnamen belegen könnte, um sich so beim Eingeben von BASIC-Programmen etwas Tipparbeit zu ersparen? Genau dazu dient die folgende Routinensammlung.

```
100 ;***********************************************************
105 ;*                                                         *
110 ;* programm: funktionstasten                               *
120 ;* routinen zur belegung der funktionstasten               *
130 ;* mit texten                                              *
```

```
140 ;*                                                        *
150 ;**********************************************************
160 ;
170 ;
180 ;
190 .ba 50108   ;*** startadresse ***
200 ;
210 ;
220 ;*** labels ***********************************************
230 .gl zp1 = 251   ;zwischenspeicher
240 .gl zp2 = 252
250 .gl zp3 = 253
260 ;
270 ;
275 ;**********************************************************
280 ;*                                                        *
285 ;* dfkey - tastentext zuweisen                            *
290 ;*                                                        *
295 ;**********************************************************
300 ;*                                                        *
305 ;* aufruf: sys 50108,(nr)=(tx$)                           *
310 ;* nr: nummer der funktionstaste (1-8)                    *
315 ;* tx$: zuweistext (maximal 10 zeichen)                   *
320 ;*                                                        *
325 ;**********************************************************
330 ;
340 dfkey       jsr $aefd        ;auf komma testen
350             jsr $aefa        ;auf 'klammer auf' pruefen
360             jsr $b79e        ;tastennummer holen
370             jsr $aef7        ;auf 'klammer auf' pruefen
380             txa              ;tastennummer in akku
390             bne dfk2         ;<>0, dann weiter
400 dfk1        jmp $b248        ;'illegal quantity' anzeigen
410 dfk2        cmp #9           ;>8?
420             bcs dfk1         ;ja, dann fehler
430             dex
440             stx zw           ;tastennr. zwischenspeichern
450             lda #$b2         ;auf '=' testen
460             jsr $aeff
470             jsr $aef1        ;tastentext in klammern holen
480             jsr $b6a3        ;stringparameter holen
490             sta zp1          ;stringlaenge zwischenspeichern
500             cmp #11          ;>10?
510             bcc dfk3         ;nein, dann weiter
520             jmp $a571        ;'string too long ' anzeigen
530 dfk3        lda zw           ;tastennummer *11
540             asl
550             asl
560             adc zw
570             asl
580             adc zw
590             clc              ;wert zu speicheranfangsadresse
600             adc #<(fkeysp)   ;addieren
```

```
610           sta zp2
620           lda #0
630           adc #>(fkeysp)
640           sta zp3
650           ldy #0            ;ausgabetext in speicher
660           lda zp1           ;stringlaenge=0?
670           bne dfk4          ;nein
680           sta (zp2),y       ;ja, dann '0' in speicher
690           rts               ;und zurueck
700 dfk4      lda ($22),y       ;zeichen aus string
710           sta (zp2),y       ;in speicher ablegen
720           iny               ;zaehler erhoehen
730           cpy zp1           ;stringlaenge erreicht?
740           bne dfk4          ;nein
750           lda #0            ;text mit null
760           sta (zp2),y       ;abschliessen
770           rts               ;fertig!
780 ;
790 ;
800 ;*******************************************************
805 ;*                                                     *
810 ;* fkeylist - aktuelle belegungen anzeigen             *
815 ;*                                                     *
820 ;*******************************************************
825 ;*                                                     *
830 ;* aufruf: sys 50196                                   *
835 ;*                                                     *
840 ;*******************************************************
845 ;
850 fkeylist  jsr $aad7         ;cr ausgeben
860           lda #<(fkeysp) ;zeiger auf textspeicher
870           sta zp1
880           lda #>(fkeysp)
890           sta zp2
900           ldx #1            ;tastennummer
910           stx zw            ;zwischenspeichern
920 fkl1      lda #"f"          ;'f' ausgeben
930           jsr $ffd2
940           lda #0            ;tastennr. (high)
950           ldx zw            ;low
960           jsr $bdcd         ;ausgeben
970           lda #":"          ;':' ausgeben
980           jsr $ffd2
990           lda #" "          ;leerstelle ausgeben
1000          jsr $ffd2
1010          ldy #0
1020          lda (zp1),y       ;kein text vorhanden?
1030          beq fkl2          ;ja, dann ausgabe ueberspringen
1040          lda zp1           ;zeiger auf textspeicher (low)
1050          ldy zp2           ;high
1060          jsr $ab1e         ;text ausgeben
1070 fkl2     jsr $aad7         ;cr ausgeben
1080          ldx zw            ;tastennummer
```

```
1090          inx             ;um 1 erhoehen
1100          cpx #9          ;>8?
1110          bcc fkl3        ;nein, dann weiter
1120          rts             ;sonst fertig!
1130 fkl3     stx zw          ;tastennr. zwischenspeichern
1140          clc             ;zeiger um 11 erhoehen
1150          lda zp1
1160          adc #11
1170          sta zp1
1180          lda zp2
1190          adc #00
1200          sta zp2
1210          jmp fkl1        ;naechsten text ausgeben
1215 ;
1220 ;
1225 ;*******************************************************
1230 ;*                                                     *
1235 ;* fkeyon - tastenbelegung aktivieren                  *
1240 ;*                                                     *
1245 ;*******************************************************
1250 ;*                                                     *
1255 ;* aufruf: sys 50275                                   *
1260 ;*                                                     *
1265 ;*******************************************************
1270 ;
1290 fkeyon   lda #<(tastabfr) ;neuen vektor
1300          sta 655          ;setzen
1310          lda #>(tastabfr)
1320          sta 656
1330          rts             ;fertig!
1340 ;
1350 ;
1355 ;*******************************************************
1360 ;*                                                     *
1365 ;* fkeyoff - tastenbelegung desaktivieren              *
1370 ;*                                                     *
1375 ;*******************************************************
1380 ;*                                                     *
1385 ;* aufruf: sys 50286                                   *
1390 ;*                                                     *
1395 ;*******************************************************
1400 ;
1410 fkeyoff  lda #<($eb48)   ;alten vektor
1420          sta 655         ;setzen
1430          lda #>($eb48)
1440          sta 656
1450          rts             ;fertig!
1455 ;
1460 ;
1465 ;*******************************************************
1470 ;*                                                     *
1475 ;* ausfuehrungsroutine                                 *
```

```
1480 ;*                                                     *
1485 ;*******************************************************
1490 ;
1495 tastabfr lda $9d        ;modus-Flag
1500          beq tt1        ;prgmodus, dann fertig
1510          ldy $cb        ;nummer der gedrueckten taste
1520          cpy $c5        ;= zuvor gedrueckter taste?
1530          beq tt1        ;ja, dann fertig
1540          lda ($f5),y    ;asc-code der taste
1550          cmp #137       ;>136?
1560          bcs tt1        ;ja, dann keine funktionstaste
1570          cmp #133       ;>=133?
1580          bcs tt2        ;ja, dann funktionstaste
1590 tt1      jmp $eb48      ;nein, dann fertig
1600 tt2      sbc #133       ;zeiger auf textspeicher
1610          sta zw         ;berechnen (tastencode * 22)
1620          asl
1630          asl
1640          adc zw
1650          asl
1660          adc zw
1670          asl
1680          ldx $028d      ;flag fuer sondertasten
1690          cpx #1         ;shift-taste gedrueckt?
1700          bne tt3        ;nein
1710          clc            ;ja, dann speicherzeiger um 11
1720          adc #11        ;erhoehen
1730 tt3      tax            ;zeiger nach x
1740          lda fkeysp,x   ;erstes zeichen holen
1750          bne tt4        ;<>0, dann weiter
1760          ldy $cb        ;asc-code der gedrueckten taste
1770          lda ($f5),y
1780          jmp $eb48      ;zur original-routine
1790 tt4      ldy #0         ;zaehler initialisieren
1800 tt5      lda fkeysp,x   ;text holen
1810          beq tt6        ;=0, dann fertig
1820          sta $0277,y    ;in tastaturpuffer schreiben
1830          inx            ;zeiger erhoehen
1840          iny
1850          jmp tt5        ;naechstes zeichen
1860 tt6      sty $c6        ;laenge des ausgabetextes
1870          ldx #$ff       ;flag 'tastaturcode ungueltig'
1880          jmp $eb26      ;zur original-routine
1890 ;
1900 ;* speicher fuer fkey-strings (laenge: 88 bytes)*******
1910 fkeysp   .by 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1920          .by 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1930          .by 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1940          .by 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Und der BASIC-Lader:

```
100 rem ******************************************************
105 rem *                                                    *
110 rem * programm: funktionstasten                          *
120 rem * programmlaenge: 357 bytes                          *
130 rem * routinen zur belegung der funktionstasten          *
140 rem * mit texten                                         *
150 rem *                                                    *
160 rem ******************************************************
165 rem *                                                    *
170 rem * dfkey - tastentext zuweisen                        *
175 rem * aufruf: sys 50108,(nr)=(tx$)                      *
180 rem * nr: nummer der funktionstaste (1-8)                *
185 rem * tx$: zuweistext (maximal 10 zeichen)               *
190 rem *                                                    *
195 rem ******************************************************
200 rem *                                                    *
205 rem * fkeylist - aktuelle belegungen anzeigen            *
210 rem * aufruf: sys 50196                                  *
215 rem *                                                    *
220 rem ******************************************************
225 rem *                                                    *
230 rem * fkeyon - tastenbelegung aktivieren                 *
235 rem * aufruf: sys 50275                                  *
240 rem *                                                    *
245 rem ******************************************************
250 rem *                                                    *
255 rem * fkeyoff - tastenbelegung desaktivieren             *
260 rem * aufruf: sys 50286                                  *
265 rem *                                                    *
270 rem ******************************************************
280 :
290 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=50108:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 32,253,174,32,250,174,32,158,183,32,1320
1010 data 247,174,138,208,3,76,72,178,201,9,1306
1020 data 176,249,202,134,2,169,178,32,255,174,1571
1030 data 32,241,174,32,163,182,133,251,201,11,1420
```

```
1040 data 144,3,76,113,165,165,2,10,10,101,789
1050 data 2,10,101,2,24,105,201,133,252,169,999
1060 data 0,105,196,133,253,160,0,165,251,208,1471
1070 data 3,145,252,96,177,34,145,252,200,196,1500
1080 data 251,208,247,169,0,145,252,96,32,215,1615
1090 data 170,169,201,133,251,169,196,133,252,162,1836
1100 data 1,134,2,169,70,32,210,255,169,0,1042
1110 data 166,2,32,205,189,169,58,32,210,255,1318
1120 data 169,32,32,210,255,160,0,177,251,240,1526
1130 data 7,165,251,164,252,32,30,171,32,215,1319
1140 data 170,166,2,232,224,9,144,1,96,134,1178
1150 data 2,24,165,251,105,11,133,251,165,252,1359
1160 data 105,0,133,252,76,35,196,169,121,141,1228
1170 data 143,2,169,196,141,144,2,96,169,72,1134
1180 data 141,143,2,169,235,141,144,2,96,165,1238
1190 data 157,240,16,164,203,196,197,240,10,177,1600
1200 data 245,201,137,176,4,201,133,176,3,76,1352
1210 data 72,235,233,133,133,2,10,10,101,2,931
1220 data 10,101,2,10,174,141,2,224,1,208,873
1230 data 3,24,105,11,170,189,201,196,208,7,1114
1240 data 164,203,177,245,76,72,235,160,0,189,1521
1250 data 201,196,240,8,153,119,2,232,200,76,1427
1260 data 181,196,132,198,162,255,76,38,235,0,1473
1270 data 0,0,0,0,0,0,0,0,0,0,0
1280 data 0,0,0,0,0,0,0,0,0,0,0
1290 data 0,0,0,0,0,0,0,0,0,0,0
1300 data 0,0,0,0,0,0,0,0,0,0,0
1310 data 0,0,0,0,0,0,0,0,0,0,0
1320 data 0,0,0,0,0,0,0,0,0,0,0
1330 data 0,0,0,0,0,0,0,0,0,0,0
1340 data 0,0,0,0,0,0,0,0,0,0,0
1350 data 0,0,0,0,0,0,0,0,0,0,0
2000 data -1:rem endmarkierung
```

Da man die Funktionstasten manchmal auch noch für andere Zwecke benötigt, gibt es die Möglichkeit, die Belegung ein- und auszuschalten. Durch SYS 50196 können Sie sich einen Überblick über die Belegung aller acht Tasten verschaffen. Eine Beispielbelegung könnte etwa so aussehen:

```
F1: LIST
F2: RUN
F3: INPUT
F4: PRINT
F5: GET
F6: GOTO
F7: GOSUB
F8: NEW
```

Noch ein kleiner Trick: Um zu erreichen, daß ein Befehl nicht nur auf den Bildschirm geschrieben, sondern gleich ausgeführt wird, hängen Sie hinten einfach ein RETURN (CHR$(13)) an:

```
SYS 50108,(1)=("LIST"+CHR$(13))
```

Wenn Sie jetzt die Funktionstaste F1 drücken, wird der LIST-Befehl sofort ausgeführt.

4.8 Ein 6510-Assembler

Der Assembler ermöglicht es uns, Maschinenprogramme genauso wie ein BASIC-Programm einzugeben. Eine Zeile besteht aus der Zeilennummer, dann kann eine Marke folgen. Nun kommt der Assembler-Befehl mit dem optionalen Operand. Damit Sie Ihre eigene auch später noch verstehen, kann, durch ein Semikolon getrennt, ein Kommentar folgen, der vom Assembler nicht beachtet wird, jedoch im Listing mit ausgegeben wird und zu Ihrer Information dient. Er entspricht also dem BASIC-Befehl REM. Eine komplette Assembler-Zeile kann z.B. so aussehen:

```
100 TEXT LDA $70,X ; STARTWERT HOLEN
```

Dieses sogenannte Quell- oder Source-Programm muß dann mit der Endung .SRC gespeichert werden. An dieser Endung kann der Assembler erkennen, daß es sich um eine Quelldatei handelt.

Der Editior

Als Editor für Quelltexte soll der normale BASIC-Editor des C64 dienen. Damit dies auch einwandfrei funktioniert, laden und starten Sie bitte vorher das folgende Programm mit RUN.

```
PROGRAMM:STARTPRG

100  FOR  I  =  53100  TO  53191
110  READ  X :  POKE  I,X :  S = S + X :  NEXT
120  DATA  169,119,160,207,141,  2,  3,140,  3,  3, 96, 32
130  DATA   96,165,134,122,132,123, 32,115,  0,170,240,243
140  DATA  162,255,134, 58,144,  6, 32,121,165, 76,225,167
150  DATA   32,107,169,160,  0,162,  0,189,  0,  2,232,201
160  DATA   32,240,248,201, 48,144,  4,201, 58,144,240,153
```

```
170  DATA    0,  2,201,  0,240,  7,189,  0,  2,200,232,208
180  DATA  242,200,200,200,200,200, 76,162,164,169,131,160
190  DATA  164,141,  2,  3,140,  3,  3, 96
200  IF  S  < >  11096  THEN   PRINT  "FEHLER IN DATAS !!" :   END
210  SYS  53100 :   PRINT  "OK"
```

Ein Beispiel:

Geben Sie das folgende Programm ein, und speichern Sie es unter dem Namen TEST.SRC.

```
100        LDX #0
110 MARKE  TXA
120        STA $0400,X
130        LDA #1
140        STA $D800,X
150        INX
160        BNE MARKE
170        RTS
180        .EN
```

Laden und starten Sie nun den Assembler. Auf dem Bildschirm sehen Sie dann:

```
          6510 - ASSEMBLER
SOURCE-FILE-NAME ? TEST  <- Hier geben Sie den Dateinamen an
LISTING  J/N     ? J
DRUCKER  J/N     ? N
```

Nach kurzer Zeit erscheint PASS 1 und dann PASS 2 auf dem Bildschirm. Jetzt fragt der Assembler, ob das erzeugte Maschinenprogramm auf Diskette gespeichert werden soll. Antworten Sie mit J(Ja).

Das fertige Programm liegt nun auf der Diskette unter dem Namen TEST.OBJ und im Speicher ab der Adresse $C000 (49152) vor. Überzeugen Sie sich, indem Sie

```
SYS 49152
```

eingeben. Doch nun noch ein paar Besonderheiten des Assemblers:

.BY Danach muß ein Wert im Bereich von 0 bis 255 folgen. Dabei können Sie außer Konstanten natürlich auch Symbole angeben, deren Wert jedoch nicht größer 255 sein darf.

```
.BY 100
.BY $7F
.BY CR
```

Bei diesem Befehl gibt es noch eine zusätzliche Option. Oft ist es nämlich erforderlich, einen 16-Bit-Wert in zwei 8-Bit-Werte zu zerlegen. Dazu gibt es die Operatoren > und <. Das Größerzeichen kennzeichnet dabei das High-Byte (Bit 7 bis 15), wärend das Kleinerzeichen für das Low-Byte steht (Bit 0-7). Dazu folgendes Beispiel:

```
100 CONST = $AB3F
110 .BY >CONST
120 .BY >CONST
```

Dieser Programmabschnitt legt die Werte $3F und $AB nacheinander im Programm ab. Die unmittelbare Adressierung erfolgt über das Zeichen #.

```
130 LDA #<CONST
140 LDX #>CONST
```

Zur Zeropage-Adressierung wird das Zeichen * benutzt.

```
100 START = $B0
110 LDA *START
```

Doch nun das Listing des Assemblers.

PROGRAMM:ASSEMBLER

```
100   REM  6510 ASSEMBLER  LE
110   PRINT  CHR$ (147):  PRINT :  PRINT :  PRINT ,"6510 - ASSEMBLER":
PRINT : DG = 8
120   INPUT "SOURCE-FILE-NAME ";SN$
130   IF  RIGHT$ (SN$,4) = ".SRC" THEN SN$ =  LEFT$ (SN$, LEN (SN$) - 4)
140  DD$ = "0":  REM  DRIVENUMMER
150   INPUT "LISTING< 2 SPACE>J/N< 5 SPACE>";A$:  IF A$ <  > "J" THEN PM
= 1:  GOTO 190
160  PF = 4: PG = 3
```

```
170  INPUT "DRUCKER< 2 SPACE>J/N< 5 SPACE>";A$:  IF A$ = "J" THEN PG =
4
180  OPEN PF,PG
190  GOSUB 5000
200 A = 0: AD = 49152:  PRINT :  PRINT : PA = A
210  PRINT "PASS 1":  GOSUB 4000:  PRINT "PASS 2": FF% = 0: FE% = 0
220 OP$ = DD$ + ": " + SN$ + ".SRC"
230  OPEN 8,DG,0,OP$
240  GET #8,A$,A$:  REM  STARTADRESSE
250  IF PM = 1 THEN  PRINT  CHR$ (145),,ZN$
260 F% = 0:  IF AD > 65535 THEN  PRINT :  PRINT :  PRINT
"BEREICHSUEBERSCHREITUNG!":  GOTO 1000
270 A = AD:  GOSUB 3240: PR$ = A$ + "< 2 SPACE>":  GOSUB 2000:  IF
LEFT$ (X$,3) = ".EN" THEN 1000
280 XX$ =  LEFT$ (X$,1):  IF XX$ = "*" THEN PR$ = "< 16 SPACE>": LN$ =
"< 5 SPACE>"
290  IF XX$ = "." OR XX$ = "*" OR XX$ = "=" THEN  GOSUB 2900:  GOTO 380
300  ON LM% GOTO 320
310 SA = OF + AD: PA = AD: LM% = 1
320 XX$ =  LEFT$ (X$,3):  FOR J = 0 TO NN%:  IF XX$ = MN$(J) THEN 350
330  NEXT
340 FL$(1) = "A": A% = 1: F% = 1:  GOSUB 1520:  GOTO 370
350  GOSUB 2400: F% = 0:  IF T% = 5 AND T%(J,9) > 0 THEN T% = 9:  REM
RELATIV
360  ON T% + 1 GOSUB
500,600,600,600,600,800,800,800,500,900,600,600,800
370  POKE OF + AD,A
380 AD = AD + A%:  IF  LEFT$ (X$,2) = "*=" THEN 400
390 LX = AD
400  REM         ************  AUSGABE
410  IF F% = 0 THEN  IF FL$(0) = " " AND FL$(1) = " " AND FL$(2) = " "
THEN 430
420 BS% = BS% + 1
430  ON PM GOTO 250
440 Y$ =  LEFT$ (Y$ + "< 11 SPACE>",11):  FOR I = 1 TO 3:  PRINT#
PF,FL$(I);:  NEXT
450  PRINT# PF,PR$ZN$LN$"< 2 SPACE>" LEFT$ (X$ + "< 4 SPACE>",6)Y$"
"RM$
460  GOTO 250
500  REM  EIN-BYTE-BEFEHLE
510 A% = 1: A = T%(J,T%):  IF A < 0 THEN FL$(2) = "A":  GOTO 1510
520  GOSUB 3240: PR$ = PR$ +  RIGHT$ (A$,2) + "< 8 SPACE>":  RETURN
600  REM  ZWEI-BYTE-BEFEHLE
610 A = T%(J,T%):  IF A < 0 THEN FL$(2) = "A":  GOTO 1500
620  GOSUB 3240: PR$ = PR$ +  RIGHT$ (A$,2)
630 YY$ = YA$:  IF  LEFT$ (YY$,1) = "#" THEN YY$ =  MID$ (YY$,2)
640  IF  LEFT$ (YY$,1) = "*" THEN YY$ =  MID$ (YY$,2)
650 A% = 2:  IF  LEFT$ (YY$,1) = ">" OR  LEFT$ (YY$,1) = "<" THEN YY$ =
MID$ (YY$,2)
660 A$ =  LEFT$ (YY$,1):  IF A$ = "$" OR A$ > "/" AND A$ < ": " THEN A$
= YY$:  GOTO 690
670 SL$ = YY$:  GOSUB 4500
680 A$ = "$" + HE$
```

```
690   GOSUB 3100
700   IF  LEFT$ (YA$,2) = "#>" THEN A =  INT (A / HI)
710   IF  LEFT$ (YA$,2) = "#<" THEN A = A -  INT (A / HI) * HI
720   IF A > LO THEN FL$(2) = "O": F% = 1: A = 0
730   GOSUB 3240:  POKE OF + AD + 1,AL%: PR$ = PR$ + " " +  RIGHT$ ("00"
+ A$,2) + "< 5 SPACE>"
740  A = T%(J,T%):  RETURN
800   REM  DREI-BYTE-BEFEHLE
810  A% = 3
820  A = T%(J,T%)
830   GOSUB 3240: PR$ = PR$ +  RIGHT$ (A$,2)
840  A$ =  LEFT$ (YA$,1):  IF A$ = "$" OR A$ > "/" AND A$ < ": " THEN A$
= YA$:  GOTO 870
850  SL$ = YA$:  GOSUB 4500
860  A$ = "$" + HE$
870   GOSUB 3100:  GOSUB 3240: PR$ = PR$ + " " +  RIGHT$ ("00" + A$,2) +
" " +  LEFT$ (A$,2) + "< 2 SPACE>"
880   POKE OF + AD + 1,AL%:  POKE OF + AD + 2,AH%
890  A = T%(J,T%):  RETURN
900   REM  RELATIV
910  A% = 2
920  A = T%(J,T%):  GOSUB 3240: PR$ = PR$ +  RIGHT$ (A$,2)
930  A$ =  LEFT$ (Y$,1):  IF A$ = "$" OR A$ > "/" AND A$ < ": " THEN A$
= Y$:  GOTO 960
940  SL$ = Y$:  GOSUB 4500
950  A$ = "$" + HE$
960   GOSUB 3100:  IF FL$(2) = "U" THEN A = AD + 2
970  DF = A - (AD + 2):  IF DF <  - 128 OR DF > 127 THEN FL$(3) = "R":
F% = 1: DF = 0
980  A = DF AND LO:  GOSUB 3240
990  PR$ = PR$ + " " +  RIGHT$ (A$,2) + "< 5 SPACE>":  POKE OF + AD +
1,A: A = T%(J,T%):  RETURN
1000  PR$ = "< 15 SPACE>":  IF F% = 0 THEN 1020
1010  BS% = BS% + 1
1020   IF AE < AD + OF THEN AE = AD + OF
1030   ON PM GOTO 1060
1040   FOR I = 0 TO 3:  PRINT# PF,FL$(I);:  NEXT
1050   PRINT# PF,PR$ZN$LN$"< 2 SPACE>" LEFT$ (X$ + "< 4 SPACE>",6)Y$"
"RM$
1060   CLOSE 8:  INPUT "AUFZEICHNUNG< 2 SPACE>J/N< 6 SPACE>";A$:  IF A$
< > "J" THEN 1130
1070  A$ = DD$ + ": " + SN$ + ".OBJ"
1080  A% =  LEN (A$):  POKE 183,A%:  POKE 187,351 AND LO:  POKE 188,351
/ HI
1090   FOR I = 1 TO A%:  POKE 350 + I, ASC ( MID$ (A$,I)):  NEXT :  REM
FILE-NAME
1100  A = SA:  GOSUB 3240:  POKE 251,AL%:  POKE 252,AH%:  REM
STARTADRESSE
1110  A = AE:  GOSUB 3240:  POKE 781,AL%:  POKE 782,AH%:  REM
ENDADRESSE
1120   POKE 780,251:  SYS 65496:  REM  SAVE
1130  A = PA:  GOSUB 3240: PA$ = A$: A = AD:  GOSUB 3240: AD$ = A$: A =
AD - PA:  GOSUB 3240
```

```
1140  BA$ = A$:  ON PM GOTO 1180
1150   PRINT# PF:  PRINT# PF,PA$" / "AD$" / "BA$
1160   PRINT# PF,"SOURCEFILE IST "SN$ + ".SRC"
1170   PRINT# PF,BS%" FEHLER":  PRINT# PF
1180   INPUT "SYMBOLTABELLE J/N ";Z$:  IF Z$ <  > "J" THEN 1400
1190  MX = 2:  IF PG = 4 THEN  PRINT# PF, CHR$ (12): MX = 5
1200   INPUT "SORTIERT< 2 SPACE>J/N ";Z$:  IF Z$ = "J" THEN 1300
1210   ON PM GOTO 1220
1220  M% = 0: P$ = "":  FOR I = LL% TO UL%
1230   IF LB$(I) = "< 5 SPACE>" THEN 1290
1240  P$ = P$ + LB$(I) + "< 2 SPACE>" + HE$(I) + "< 5 SPACE>": M% = M% +
1
1250   IF M% <  > MX THEN 1290
1260   ON PM GOTO 1280
1270   PRINT# PF,P$
1280  P$ = "": M% = 0:  IF I >  = UL% THEN 1400
1290   NEXT I:  IF P$ <  > "" THEN 1260
1300  HI$ =  CHR$ (127) +  CHR$ (127) +  CHR$ (127) +  CHR$ (127) +
CHR$ (127): F% = 0:  REM  SORT
1310  M% = 0: SL$ = HI$:  FOR I = LL% TO UL%:  IF LB$(I) = "< 5 SPACE>"
THEN 1340
1320   IF LB$(I) < SL$ THEN SL$ = LB$(I): M% = I + 1
1330  UL% = I
1340   NEXT I:  IF F% < MX THEN 1360
1350  F% = 0:  IF PM = 0 THEN  PRINT# PF
1360   IF M% = 0 THEN 1400
1370   ON PM GOTO 1390
1380   PRINT# PF,SL$"< 2 SPACE>"HE$(M% - 1)"< 5 SPACE>";
1390  LB$(M% - 1) = "< 5 SPACE>": F% = F% + 1:  GOTO 1310
1400   REM
1410   IF PG = 4 THEN  PRINT# PF, CHR$ (12)
1420   CLOSE PF:  END
1500   POKE OF + AD + 2,0:  REM  NOP-FUELLER
1510   POKE OF + AD + 1,0
1520  A = 0: PR$ = PR$ + NP$(A%):  RETURN
1600   IF  LEFT$ (LN$,1) = "." THEN I =  - 1:  RETURN
1610   IF  MID$ (LN$,4,1) <  > " " THEN I = NN% + 1:  RETURN
1620  MN$ =  LEFT$ (LN$,3):  REM  LABEL = MNEMONIC ?
1630   FOR I = 0 TO NN%:  IF MN$ <  > MN$(I) THEN  NEXT
1640   RETURN
2000   GET #8,A$,B$:  IF A$ + B$ = "" THEN 2290:  REM  LINKADRESSE
2010   GET #8,Z1$,Z2$
2020  ZN =  ASC (Z1$ +  CHR$ (0)) + HI *  ASC (Z2$ +  CHR$ (0))
2030  ZN$ =  RIGHT$ ("< 3 SPACE>" +  STR$ (ZN),5) + " "
2040   GOSUB 2300:  IF FF% THEN  RETURN
2050  LN$ = "": X$ = "": Y$ = "": RM$ = "": X% = 0
2060   FOR I = 0 TO 3: FL$(I) = " ":  NEXT I:  IF Z$ = "*" THEN 2190
2070   IF Z$ = ";" THEN 2280
2080   REM  LABELNAME
2090   IF Z$ = " " OR FF% THEN LN$ =  LEFT$ (LN$ + "< 4 SPACE>",5):
GOTO 2120
2100  LN$ = LN$ + Z$:  IF  LEN (LN$) = 6 THEN X% = 1: FL$(0) = "L"
2110   GOSUB 2300:  GOTO 2090
```

```
2120  GOSUB 1600:  IF I <  = NN% THEN X$ = LN$: LN$ = "< 5 SPACE>":
GOTO 2200
2130 X% =  ASC (LN$):  IF X% < 65 OR X% > 90 THEN FL$(0) = "S"
2140  REM  OPERATION
2150  GOSUB 2300:  IF FF% THEN  RETURN
2160  IF Z$ <  > " " THEN 2190
2170  GOTO 2150
2180  GOSUB 2300:  IF FF% THEN  RETURN
2190  IF Z$ <  > " " THEN X$ = X$ + Z$:  GOTO 2180
2200  IF FF% THEN  RETURN
2210  IF Z$ = ";" THEN 2280
2220  IF Z$ <  > " " THEN 2260:  REM  OPERAND
2230  GOSUB 2300:  IF FF% THEN  RETURN
2240  GOTO 2200
2250  GOSUB 2300:  IF FF% THEN  RETURN
2260  IF Z$ <  > " " THEN Y$ = Y$ + Z$:  GOTO 2250
2270  GOSUB 2300:  IF FF% THEN  RETURN :  REM  BEMERKUNG
2280 RM$ = RM$ + Z$:  GOTO 2270
2290 X$ = ".EN": RM$ = "END ASSUMED": LN$ = "< 6 SPACE>": Y$ = "": ZN$
= "< 6 SPACE>":  RETURN
2300  GET #8,Z$: FF% = Z$ = "":  RETURN
2400  REM  ADRESSIERUNGSART ERMITTLEN
2410  IF Y$ = "" THEN T% = 8:  RETURN :  REM  IMPLIZIT
2420 YA$ = Y$:  IF  LEFT$ (YA$,1) = "(" THEN YA$ =  MID$ (YA$,2)
2430  IF  RIGHT$ (YA$,1) = ")" THEN YA$ =  LEFT$ (YA$, LEN (YA$) - 1)
2440  IF  RIGHT$ (YA$,3) = "),Y" THEN YA$ =  LEFT$ (YA$, LEN (YA$) - 3)
2450  IF  RIGHT$ (YA$,2) = ",Y" OR  RIGHT$ (YA$,2) = ",X" THEN YA$ =
LEFT$ (YA$, LEN (YA$) - 2)
2460 Z$ = Y$: K$ =  LEFT$ (Y$,1)
2470  IF Z$ = "A" THEN T% = 0:  RETURN :  REM  AKKU
2480  IF K$ = "#" THEN T% = 1:  RETURN :  REM  IMMIDIATE
2490  IF K$ = "(" THEN 2600:  REM  INDIREKT
2500 ZP = K$ = "*":  REM  ZEROPAGE
2510 Z$ =  MID$ (Y$,2 + ZP)
2520  IF  LEN (Z$) < 2 THEN 2550
2530 K$ =  MID$ (Z$, LEN (Z$) - 1,1)
2540  IF K$ = "," THEN 2570:  REM  INDIZIERT
2550 T% = 5
2560 T% = T% + 3 * ZP:  RETURN :  REM  ABSOLUT BZW. ZEROPAGE
2570 K$ =  RIGHT$ (Z$,1):  IF K$ = "X" THEN T% = 6:  GOTO 2560
2580  IF K$ = "Y" THEN T% = 7:  GOTO 2560
2590 T% =  - 1:  RETURN : REM SYNTAX ERROR
2600 K$ =  RIGHT$ (Z$,1):  IF K$ = ")" THEN 2630
2610  IF  RIGHT$ (Z$,2) <  > ",Y" THEN 2590
2620 T% = 11:  RETURN
2630  IF  MID$ (Z$, LEN (Z$) - 2,2) = ",X" THEN T% = 10:  RETURN
2640 T% = 12:  RETURN
2700  IF X$ = "=" THEN 2730:  REM  PSEUDO-OPS PASS 1
2710  IF  LEFT$ (X$,2) = "*=" THEN 2780
2720 A% = 0:  RETURN
2730 A% = 0:  IF Y$ = "*" THEN  RETURN
2740 A% =  ASC ( LEFT$ (LN$,1)):  IF A% < 65 OR A% > 90 THEN  RETURN
```

```
2750  A$ =  LEFT$ (Y$,1):  IF A$ <  > "$" AND (A$ < "0" OR A$ > "9")
THEN  RETURN
2760  A$ = Y$:  GOSUB 3100:  IF F% THEN ML$(HC%) = FL$(2):  RETURN
2770   GOSUB 3240: HE$(HC%) =  RIGHT$ ("0000" + A$,4):  RETURN
2780  A% = 0: Y1$ =  LEFT$ (Y$,1):  IF Y1$ = "$" OR Y1$ > "/" AND Y1$ <
": " THEN 2800
2790   RETURN
2800  A$ = Y$:  GOSUB 3100:  IF F% THEN  RETURN
2810  HA = A:  GOSUB 3240: X% =  ASC ( LEFT$ (LN$ +  CHR$ (0),1)):  IF
X% > 64 AND X% < 91 THEN HE$(HC%) = A$
2820   RETURN
2900   IF XX$ = "=" THEN 2940:  REM  PSEUDO-OPS PASS 2
2910   IF  LEFT$ (X$,2) = "*=" THEN 2990
2920  FL$(1) = "S"
2930  A% = 0: F% = 1: PR$ = "< 16 SPACE>":  RETURN
2940  A% = 0
2950  A$ =  LEFT$ (Y$,1)
2960   IF A$ <  > "*" AND A$ <  > "$" AND (A$ < "0" OR A$ > "9") THEN
FL$(2) = "S":  GOTO 2930
2970  SL$ = LN$: F% = 0:  GOSUB 4500:  IF F% THEN FL$(0) = FL$(2):
FL$(2) = " ":  GOTO 2930
2980  PR$ = HE$ + "< 12 SPACE>":  RETURN
2990  A% = 0: Y1$ =  LEFT$ (Y$,1):  IF Y1$ = "$" OR Y1$ > "/" AND Y1$ <
": " THEN 3010
2991  YZ$ =  LEFT$ (Y$,1): LH% = YZ$ = ">" OR YZ$ = ">" OR YZ$ = "<":
YA$ =  MID$ (Y$,1 - LH%)
2992  YZ$ =  LEFT$ (YA$,1):  IF YZ$ = "$" OR YZ$ > "/" AND YZ$ < ": "
THEN HE$ = YA$:  GOTO 2994
2993  SL$ = YA$: F% = 0:  GOSUB 4500: HE$ = "$" + HE$:  IF F% THEN
FL$(0) = FL$(2): FL$(2) = " "
2994  A$ = HE$:  GOSUB 3100:  IF A > LO AND LH% = 0 THEN A = 0: FL$(1) =
"O"
2995   IF  LEFT$ (Y$,1) = ">" THEN A =  INT (A / HI)
2996   IF  LEFT$ (Y$,1) = "<" THEN A = A -  INT (A / HI) * HI
2998   POKE AD,A: A% = 1:  GOSUB 3240: PR$ = PR$ +  RIGHT$ ("00" + A$,2)
+ "< 6 SPACE>":  RETURN
3000  FL$(2) = "S": F% = 1:  GOTO 3030
3010  A$ = Y$:  GOSUB 3100:  IF F% THEN 3030
3020  AD = A:  GOSUB 3240: PR$ = A$ + "< 2 SPACE>"
3030  PR$ = PR$ + "< 10 SPACE>":  RETURN
3100   REM  WANDLUNG HEX -> DEC  A$ -> A
3110  Z$ =  LEFT$ (A$,1):  IF Z$ = "$" THEN A$ =  RIGHT$ (A$, LEN (A$) -
1):  GOTO 3150
3120   IF Z$ < "0" OR Z$ > "9" THEN FL$(2) = "S": F% = 1:  RETURN
3130  A =  VAL (A$):  IF A > 65535 OR A < 0 THEN FL$(2) = "O": F% = 1
3140   RETURN
3150  A = 0: L% =  LEN (A$):  IF L% > 4 THEN F% = 1: FL$(2) = "L":
RETURN
3200   FOR I = 1 TO L%: AA% =  ASC ( MID$ (A$,I)) - 48
3210   IF AA% < 0 OR AA% > 9 THEN  IF AA% < 17 OR AA% > 22 THEN F% = 1:
FL$(2) = "S":  RETURN
3220   IF AA% > 9 THEN AA% = AA% - 7
3230  A = A + AA% * 16 ^ (L% - I):  NEXT :  RETURN
```

```
3240 AH% = A / HI: AL% = A - AH% * HI: A$ = A$(AH% / 16) + A$(AH% AND
15) + A$(AL% / 16) + A$(AL% AND 15)
3250  RETURN
4000  DIM LB$(349),HE$(349),ML$(349): HA = AD:  REM  ADRESSBUCH
AUFBAUEN
4010  FOR I = 0 TO 349: LB$(I) = "< 5 SPACE>": HE$(I) = "0000": ML$(I)
= " ":  NEXT
4020 OP$ = DD$ + ": " + SN$ + ".SRC"
4030  OPEN 8,DG,0,OP$
4040  GET #8,A$,A$: LL% = 349
4050  IF ST <  > 0 THEN  CLOSE 8:  END
4060  GOSUB 2000:  PRINT  CHR$ (145),ZN$:  IF LN$ = "" OR  LEFT$
(LN$,1) = " " THEN 4210
4070 X% =  ASC ( LEFT$ (LN$,1)):  IF X% < 65 OR X% > 90 THEN 4210
4080  GOSUB 4100:  GOTO 4130
4090 LN$ =  LEFT$ (LN$ + "< 4 SPACE>",5):  REM  HASH-CODE BILDEN
4100 HC = 0:  FOR I = 1 TO 5
4110 HC% =  ASC ( MID$ (LN$,I,1)): HC = HC + (HC% / 10 -  INT (HC% /
10)) * 10 ^ (6 - I):  NEXT I
4120 HC% = (HC / 307 -  INT (HC / 307)) * 300:  RETURN
4130 A = HA:  GOSUB 3240
4140  IF LB$(HC%) <  > "< 5 SPACE>" THEN 4180
4150 LB$(HC%) = LN$: HE$(HC%) = A$:  IF HC% > UL% THEN UL% = HC%
4160  IF HC% < LL% THEN LL% = HC%
4170  GOTO 4210
4180  IF LB$(HC%) = LN$ THEN ML$(HC%) = "M":  GOTO 4210
4190 HC% = HC% + 1:  IF HC% < 350 THEN 4140
4200  PRINT "SYMBOLTABELLE VOLL":  CLOSE 8:  END
4210  IF X$ = ".EN" THEN  CLOSE 8:  RETURN
4220 XX$ =  LEFT$ (X$,1):  IF XX$ = "." OR XX$ = "*" OR XX$ = "=" THEN
GOSUB 2700: HA = HA + A%:  GOTO 4060
4230 F% = 0: XX$ =  LEFT$ (X$,3):  FOR J = 0 TO NN%:  IF XX$ <  >
MN$(J) THEN  NEXT :  GOTO 4270
4240  GOSUB 2400
4250  IF T%(J,T%) >  = 0 THEN 4280
4260  IF T% = 5 AND T%(J,9) >  = 0 THEN T% = 9:  GOTO 4280
4270 F% = 1: HA = HA + 1:  GOTO 4060
4280 HA = HA + L%(T%):  GOTO 4060
4500  REM             *********  LABEL SUCHEN
4510 X% =  ASC ( LEFT$ (SL$,1)):  IF X% < 65 OR X% > 90 THEN FL$(2) =
"S": F% = 1: HE$ = "0000":  RETURN
4520  IF  LEN (SL$) > 5 THEN FL$(2) = "L"
4530 SV$ = LN$: LN$ = SL$:  GOSUB 4090: SL$ = LN$: LN$ = SV$
4540  IF LB$(HC%) = "< 5 SPACE>" OR HC% > UL% THEN FL$(2) = "U": F% =
1: HE$ = "0000":  RETURN
4550  IF LB$(HC%) <  > SL$ THEN 4580
4560 HE$ = HE$(HC%):  IF ML$(HC%) <  > " " THEN FL$(2) = ML$(HC%)
4570  RETURN
4580 HC% = HC% + 1:  GOTO 4540
4590 Y1$ = "": Y2$ = "": I = 1:  REM  ZERLEGEN VON Y$ IN Y1$ UND Y2$
4600  IF  MID$ (Y$,I,1) <  > "," THEN Y1$ = Y1$ +  MID$ (Y$,I,1)
4610  IF I >  LEN (Y$) THEN F% = 1:  RETURN
4620  IF  MID$ (Y$,I,1) <  > "," THEN I = I + 1:  GOTO 4600
```

```
4630  I = I + 1:  IF I >  LEN (Y$) THEN F% = 1:  RETURN
4640  Y2$ = Y2$ +  MID$ (Y$,I,1):  IF I =  LEN (Y$) THEN F% = 0:  RETURN
4650  I = I + 1:  GOTO 4640
5000   READ NN%: HI = 256: LO = 255
5010   DIM A$(15),MN$(NN%),T%(NN%,12),L%(12),FL$(3),NP$(3)
5020   FOR I = 0 TO 15:  READ A$(I):  NEXT
5030  NP$(1) = "00< 8 SPACE>": NP$(2) = "00 00< 5 SPACE>": NP$(3) = "00
00 00< 2 SPACE>"
5040   FOR I = 0 TO 12:  READ L%(I):  NEXT
5050   FOR J = 0 TO NN%:  READ MN$(J):  FOR JJ = 0 TO 12:  READ A$:  IF
A$ = "-1" THEN A =  - 1:  GOTO 5070
5060  A = 0:  FOR I = 1 TO 2: X =  ASC ( RIGHT$ (A$,I)) - 48: X = X + (X
> 9) * 7: A = A + X * 16 ^ (I - 1):  NEXT
5070  T%(J,JJ) = A:  NEXT :  NEXT :  RETURN
6000   DATA  55 :  REM  ANZAHL MNEMONICS
6010   DATA  0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
6020   DATA  1,2,2,2,2,3,3,3,1,2,2,2,3
7000   DATA  ADC,-1,69,65,75,-1,6D,7D,79,-1,-1,61,71,-1
7010   DATA  AND,-1,29,25,35,-1,2D,3D,39,-1,-1,21,31,-1
7020   DATA  ASL,0A,-1,06,16,-1,0E,1E,-1,-1,-1,-1,-1,-1
7030   DATA  BCC,-1,-1,-1,-1,-1,-1,-1,-1,-1,90,-1,-1,-1
7040   DATA  BCS,-1,-1,-1,-1,-1,-1,-1,-1,-1,B0,-1,-1,-1
7050   DATA  BEQ,-1,-1,-1,-1,-1,-1,-1,-1,-1,F0,-1,-1,-1
7060   DATA  BMI,-1,-1,-1,-1,-1,-1,-1,-1,-1,30,-1,-1,-1
7070   DATA  BIT,-1,-1,24,-1,-1,2C,-1,-1,-1,-1,-1,-1,-1
7080   DATA  BNE,-1,-1,-1,-1,-1,-1,-1,-1,-1,D0,-1,-1,-1
7090   DATA  BPL,-1,-1,-1,-1,-1,-1,-1,-1,-1,10,-1,-1,-1
7100   DATA  BRK,-1,-1,-1,-1,-1,-1,-1,-1,00,-1,-1,-1,-1
7110   DATA  BVC,-1,-1,-1,-1,-1,-1,-1,-1,-1,50,-1,-1,-1
7120   DATA  BVS,-1,-1,-1,-1,-1,-1,-1,-1,-1,70,-1,-1,-1
7130   DATA  CLC,-1,-1,-1,-1,-1,-1,-1,-1,18,-1,-1,-1,-1
7140   DATA  CLD,-1,-1,-1,-1,-1,-1,-1,-1,D8,-1,-1,-1,-1
7150   DATA  CLI,-1,-1,-1,-1,-1,-1,-1,-1,58,-1,-1,-1,-1
7160   DATA  CLV,-1,-1,-1,-1,-1,-1,-1,-1,B8,-1,-1,-1,-1
7170   DATA  CMP,-1,C9,C5,D5,-1,CD,DD,D9,-1,-1,C1,D1,-1
7180   DATA  CPX,-1,E0,E4,-1,-1,EC,-1,-1,-1,-1,-1,-1,-1
7190   DATA  CPY,-1,C0,C4,-1,-1,CC,-1,-1,-1,-1,-1,-1,-1
7200   DATA  DEC,-1,-1,C6,D6,-1,CE,DE,-1,-1,-1,-1,-1,-1
7210   DATA  DEX,-1,-1,-1,-1,-1,-1,-1,-1,CA,-1,-1,-1,-1
7220   DATA  DEY,-1,-1,-1,-1,-1,-1,-1,-1,88,-1,-1,-1,-1
7230   DATA  EOR,-1,49,45,55,-1,4D,5D,59,-1,-1,41,51,-1
7240   DATA  INC,-1,-1,E6,F6,-1,EE,FE,-1,-1,-1,-1,-1,-1
7250   DATA  INX,-1,-1,-1,-1,-1,-1,-1,-1,E8,-1,-1,-1,-1
7260   DATA  INY,-1,-1,-1,-1,-1,-1,-1,-1,C8,-1,-1,-1,-1
7270   DATA  JMP,-1,-1,-1,-1,-1,4C,-1,-1,-1,-1,-1,-1,6C
7280   DATA  JSR,-1,-1,-1,-1,-1,20,-1,-1,-1,-1,-1,-1,-1
7290   DATA  LDA,-1,A9,A5,B5,-1,AD,BD,B9,-1,-1,A1,B1,-1
7300   DATA  LDX,-1,A2,A6,-1,B6,AE,-1,BE,-1,-1,-1,-1,-1
7310   DATA  LDY,-1,A0,A4,B4,-1,AC,BC,-1,-1,-1,-1,-1,-1
7320   DATA  LSR,4A,-1,46,56,-1,4E,5E,-1,-1,-1,-1,-1,-1
7330   DATA  NOP,-1,-1,-1,-1,-1,-1,-1,-1,EA,-1,-1,-1,-1
7340   DATA  ORA,-1,09,05,15,-1,0D,1D,19,-1,-1,01,11,-1
7350   DATA  PHA,-1,-1,-1,-1,-1,-1,-1,-1,48,-1,-1,-1,-1
```

```
7360  DATA  PHP,-1,-1,-1,-1,-1,-1,-1,-1,08,-1,-1,-1,-1
7370  DATA  PLA,-1,-1,-1,-1,-1,-1,-1,-1,68,-1,-1,-1,-1
7380  DATA  PLP,-1,-1,-1,-1,-1,-1,-1,-1,28,-1,-1,-1,-1
7390  DATA  ROL,2A,-1,26,36,-1,2E,3E,-1,-1,-1,-1,-1,-1
7400  DATA  ROR,6A,-1,66,76,-1,6E,7E,-1,-1,-1,-1,-1,-1
7410  DATA  RTI,-1,-1,-1,-1,-1,-1,-1,-1,40,-1,-1,-1,-1
7420  DATA  RTS,-1,-1,-1,-1,-1,-1,-1,-1,60,-1,-1,-1,-1
7430  DATA  SBC,-1,E9,E5,F5,-1,ED,FD,F9,-1,-1,E1,F1,-1
7440  DATA  SEC,-1,-1,-1,-1,-1,-1,-1,-1,38,-1,-1,-1,-1
7450  DATA  SED,-1,-1,-1,-1,-1,-1,-1,-1,F8,-1,-1,-1,-1
7460  DATA  SEI,-1,-1,-1,-1,-1,-1,-1,-1,78,-1,-1,-1,-1
7470  DATA  STA,-1,-1,85,95,-1,8D,9D,99,-1,-1,81,91,-1
7480  DATA  STX,-1,-1,86,-1,96,8E,-1,-1,-1,-1,-1,-1,-1
7490  DATA  STY,-1,-1,84,94,-1,8C,-1,-1,-1,-1,-1,-1,-1
7500  DATA  TAX,-1,-1,-1,-1,-1,-1,-1,-1,AA,-1,-1,-1,-1
7510  DATA  TAY,-1,-1,-1,-1,-1,-1,-1,-1,A8,-1,-1,-1,-1
7520  DATA  TSX,-1,-1,-1,-1,-1,-1,-1,-1,BA,-1,-1,-1,-1
7530  DATA  TXA,-1,-1,-1,-1,-1,-1,-1,-1,8A,-1,-1,-1,-1
7540  DATA  TXS,-1,-1,-1,-1,-1,-1,-1,-1,9A,-1,-1,-1,-1
7550  DATA  TYA,-1,-1,-1,-1,-1,-1,-1,-1,98,-1,-1,-1,-1
```

4.9 Ein Einzelschrittsimulator für den 6510

In diesem Unterkapitel finden Sie ein Programm, das beim Austesten Ihrer Programme - sowie der Fehlersuche - sehr nützlich sein kann. Dieses Programm simuliert die Arbeit eines 6510-Prozessors. Wenn Sie das Programm mit RUN starten, erscheinen die Register-Bezeichnungen des Prozessors sowie anschließend die Inhalte der Register auf dem Bildschirm:

```
 PC    AC XR YR SR SP  NV-BDIZC
0000   00 00 00 00 00  00100000
```

Die Bedeutung haben Sie sicherlich schon erkannt:

```
PC  Programmzähler (program counter)
AC  Akkumulator    (accu)
XR  X-Register
YR  Y-Register
SR  Status-Register
SP  Stapelzeiger  (stack pointer)

N   Negative-Flag
V   Overflow-Flag
B   Break-Flag
D   Dezimal-Flag
I   Interrupt-Flag
```

```
Z  Zero-Flag
C  Carry-Flag
```

Die oben angegebenen Werte sind die Startwerte, die durch Drücken einer Taste geändert werden können.

Sie können durch Drücken der folgenden Tasten die Register-Inhalte ändern.

```
E  Programmzähler
A  Accu
X  X-Register
Y  Y-Register
S  Stack-Pointer
N  Negative-Flag
V  Overflow-Flag
B  Break-Flag
D  Dezimal-Flag
I  Interrupt-Flag
Z  Zero-Flag
C  Carry-Flag
```

Die wichtigste Taste jedoch ist die Leertaste. Betätigen Sie diese Taste, so wird der Maschinenbefehl der sich gerade im Programmzähler befindet und in disassemblierter Form unterhalb der Register zusehen ist, ausgeführt. Danach können sich auch die Register- und Flag-Inhalte ändern.

Ein Beispiel: Benutzen Sie dazu der Einfachheit halber einen Programmabschnitt aus dem Betriebssystem. Dazu setzen Sie den Progammzähler auf $A81D. Sie erhalten dann folgende Anzeige:

```
 PC   AC XR YR SR SP  NV-BDIZC
A81D  00 00 00 20 FF  00100000

A81D        SEC
```

Drücken Sie nun die Leertaste, um die Ausführung des Befehls zu simulieren, so erhalten Sie:

```
 PC   AC XR YR SR SP  NV-BDIZC
A81D  00 00 00 21 FF  00100001

A81D        LDA $2B
```

Der Befehl wurde also ausgeführt, das Carry-Flag ist gesetzt. Der Wert des Status-Registers hat sich automatisch auf $21 mitgeändert. Der Programmzähler wurde um eins auf $A81E weitergezählt. Dort steht nun der Ladebefehl. Auch diesen können Sie nun durch Drücken der Leertaste ausführen lassen.

```
 PC   AC XR YR SR SP  NV-BDIZC
A820  01 00 00 21 FF  00100001

A820          SBC #$01
```

Der Akkumulator wurde also mit dem Inhalt der Speicherstelle $2b geladen, die den Wert 1 enthielt. Beachten Sie bitte das Z- und N-Flag gelöscht bleiben. Der Programmzähler steht nun auf $A820, also zwei Bytes weiter.

An dieser Stelle möchte ich die Simulation abbrechen. Der große Vorteil dieses Simulators ist, daß Sie genau sehen, was bei jedem Befehl passiert. Sie können vor jeder Ausführung eines Befehls willkürlich Register- und Flag-Inhalte ändern, um zu sehen, wie der Prozessor darauf reagiert und nach der Ausführung des Befehls den Programmzähler wieder auf diesen zurücksetzen, um ihn mit geänderten Register- oder Flag-Inhalten noch einmal ausführen zu lassen. Hier nun das Programm:

PROGRAMM:SIMULATOR

```
100   PRINT "<CLR HOME><CTRL H><WHT><CRSR DOWN>< 6 SPACE>6510
EINZELSCHRITT-SIMULATOR"
110   PRINT "< 6 SPACE>--------------------------
120   PRINT "< 4 SPACE><COMM A>< 4 SHIFT *><COMM R>< 16 SHIFT *><COMM
R>< 8 SHIFT *><COMM S>
130   PRINT "< 4 SPACE><SHIFT -> PC <SHIFT -> AC XR YR SR SP <SHIFT -
>NV-BDIZC<SHIFT ->"
140   PRINT "< 4 SPACE><SHIFT ->< 4 SPACE><SHIFT ->< 16 SPACE><SHIFT ->< 
8 SPACE><SHIFT ->"
150   PRINT "< 4 SPACE><COMM Z>< 4 SHIFT *><COMM E>< 16 SHIFT *><COMM
E>< 8 SHIFT *><COMM X>
160  FF = 255: HI = 256: UL = 2 ^ 16: SC = 2 ^ 15 - 1: SP = FF
170   DIM MN$(FF),OP(FF),AD(FF),SP(FF),H$(15)
180   FOR J = 0 TO 15:  READ H$(J):  NEXT
190   FOR J = 0 TO FF:  READ MN$(J),OP(J),AD(J):  NEXT
200   REM  REGISTER-ANZEIGE
210   PRINT "<HOME>< 5 CRSR DOWN>< 5 CRSR RIGHT>";
215   IF PC >  = UL THEN PC = PC - UL
220  A = PC:  GOSUB 2290:  PRINT "< 2 CRSR RIGHT>";
```

```
230  A = AC:  GOSUB 2320:  PRINT "<CRSR RIGHT>";
240  A = XR:  GOSUB 2320:  PRINT "<CRSR RIGHT>";
250  A = YR:  GOSUB 2320:  PRINT "<CRSR RIGHT>";
255  GOSUB 900:  REM  SR
260  A = SR:  GOSUB 2320:  PRINT "<CRSR RIGHT>";
270  A = SP:  GOSUB 2320:  PRINT "< 2 CRSR RIGHT>";
280  PRINT  CHR$ (48 + N);
290  PRINT  CHR$ (48 + V);
300  PRINT "1";
310  PRINT  CHR$ (48 + B);
320  PRINT  CHR$ (48 + D);
330  PRINT  CHR$ (48 + I);
340  PRINT  CHR$ (48 + Z);
350  PRINT  CHR$ (48 + C)
360  PRINT "< 5 CRSR DOWN>< 19 SPACE>< 18 CRSR LEFT>";
400  GET T$:  IF T$ = "" THEN 400
405  IF T$ = " " THEN 1100:  REM  SIMULATION
410  IF T$ = "P" THEN  PRINT "PC< 2 SPACE>";: A = PC:  GOSUB 2290:
INPUT "< 6 CRSR LEFT>";A$:  GOSUB 2380: PC = A:  GOTO 1000
420  IF T$ = "A" THEN T$ = "AC": A = AC:  GOSUB 540: AC = A:  GOTO 200
430  IF T$ = "X" THEN T$ = "XR": A = XR:  GOSUB 540: XR = A:  GOTO 200
440  IF T$ = "Y" THEN T$ = "YR": A = YR:  GOSUB 540: YR = A:  GOTO 200
450  IF T$ = "S" THEN T$ = "SP": A = SP:  GOSUB 540: SP = A:  GOTO 200
460  IF T$ = "N" THEN N = 1 - N:  GOTO 200
470  IF T$ = "V" THEN V = 1 - V:  GOTO 200
480  IF T$ = "B" THEN B = 1 - B:  GOTO 200
490  IF T$ = "D" THEN D = 1 - D:  GOTO 200
500  IF T$ = "I" THEN I = 1 - I:  GOTO 200
510  IF T$ = "Z" THEN Z = 1 - Z:  GOTO 200
520  IF T$ = "C" THEN C = 1 - C:  GOTO 200
525  IF T$ = "<CRSR DOWN>" THEN S = P: E = P: PC = P:  GOTO 1010
527  IF T$ = "M" THEN 3000
528  IF T$ = "E" THEN 3100
530  GOTO 400
540  PRINT T$"< 2 SPACE>";:  GOSUB 2320:  INPUT "< 4 CRSR LEFT>";A$:
GOTO 2380
900  SR = N * 128 + V * 64 + 32 + B * 16 + D * 8 + I * 4 + Z * 2 + C:
RETURN
910  N =  SGN (SR AND 128): V =  SGN (SR AND 64): B =  SGN (SR AND 16):
D =  SGN (SR AND 8)
920  I =  SGN (SR AND 4): Z =  SGN (SR AND 2): C = SR AND 1:  RETURN
980  N =  SGN (AC AND 128): Z = 1 -  SGN (AC):  REM  FLAGS
990  PC = PC + 1 + L
1000  S = PC: E = PC
1010  PRINT "<HOME>< 8 CRSR DOWN>":  GOSUB 2040:  GOTO 200
1100  A = OP( PEEK (PC)): L = 0:  IF A = 0 THEN 990
1110  ON A GOTO
1200,1210,1220,1230,1240,1250,1260,1270,1280,1290,1300,1310,1320,1330
1115  A = A - 14
1120  ON A GOTO
1340,1350,1360,1370,1380,1390,1400,1410,1420,1430,1440,1450,1460,1470
1125  A = A - 14
```

```
1130  ON A GOTO
1480,1490,1500,1510,1520,1530,1540,1550,1560,1570,1580,1590,1600,1610
1135  A = A - 14
1140  ON A GOTO
1620,1630,1640,1650,1660,1670,1680,1690,1700,1710,1720,1730,1740,1750
1150  GOTO 200
1200  IF D THEN 1205:  REM  ADC
1201  GOSUB 1900: V = 1 -  SGN (AC AND 128): AC = AC + OP + C: C =  -
(AC > FF)
1202  AC = AC AND FF: N =  SGN (AC AND 128): V = V AND N:  GOTO 980
1205  GOSUB 1900: AC =  VAL (H$(AC / 16) + H$(AC AND 15)): OP =  VAL
(H$(OP / 16) + H$(OP AND 15))
1206  AC = AC + OP + C: C =  - (AC > 99):  IF AC > 99 THEN AC = AC - 100
1207  A$ =  MID$ ( STR$ (AC),2):  GOSUB 2390: AC = A:  GOTO 980
1210  REM  AND
1211  GOSUB 1900: AC = AC AND OP:  GOTO 980
1220  REM  ASL
1221  IF AD( PEEK (PC)) = 4 THEN AC = AC * 2: C =  - (AC > FF): AC = AC
AND FF:  GOTO 980
1222  GOSUB 1900: A = OP * 2: C =  - (A > FF): A = A AND FF:  GOSUB
1850
1223  N =  SGN (OP AND FF): Z = 1 -  SGN (OP):  GOTO 990
1230  REM  BCC
1231  FL = 1 - C:  GOTO 1800
1240  REM  BCS
1241  FL = C:  GOTO 1800
1250  REM  BEQ
1251  FL = Z:  GOTO 1800
1260  REM  BIT
1261  GOSUB 1900: N =  SGN (OP AND 128): V =  SGN (OP AND 64): Z = 1 -
SGN (OP AND AC):  GOTO 990
1270  REM  BMI
1271  FL = N:  GOTO 1800
1280  REM  BNE
1281  FL = 1 - Z:  GOTO 1800
1290  REM  BPL
1291  FL = 1 - N:  GOTO 1800
1300  REM  BRK
1301  PC = PC + 2:  IF PC >  = UL THEN PC = PC - UL
1302  PH =  INT (PC / HI): PL = PC - PH * HI: SP(SP) = PH: SP = SP - 1
AND FF: SP(SP) = PL: SP = SP - 1 AND FF
1303  SP(SP) = SR: SP = SP - 1 AND FF: B = 1: I = 1:  GOSUB 900: PC =
PEEK (UL - 2) + HI *  PEEK (UL - 1):  GOTO 1000
1310  REM  BVC
1311  FL = 1 - V:  GOTO 1800
1320  REM  BVS
1321  FL = V:  GOTO 1800
1330  REM  CLC
1331  C = 0:  GOTO 990
1340  REM  CLD
1341  D = 0:  GOTO 990
1350  REM  CLI
1351  I = 0:  GOTO 990
```

```
1360  REM  CLV
1361  V = 0:  GOTO 990
1370  REM  CMP
1371  GOSUB 1900: A = AC - OP
1372  N =  SGN (A AND 128): Z =  - (A = 0): C =  - (A >  = 0):  GOTO 990
1380  REM  CPX
1381  GOSUB 1900: A = XR - OP:  GOTO 1372
1390  REM  CPY
1391  GOSUB 1900: A = YR - OP:  GOTO 1372
1400  REM  DEC
1401  GOSUB 1900: A = OP - 1 AND FF:  GOSUB 1850
1402  GOTO 1442
1410  REM  DEX
1411  XR = (XR - 1) AND FF:  GOTO 1452
1420  REM  DEY
1421  YR = (YR - 1) AND FF:  GOTO 1462
1430  REM  EOR
1431  GOSUB 1900: AC = (AC OR OP) AND  NOT (AC AND OP)
1432  GOTO 980
1440  REM  INC
1441  GOSUB 1900: A = OP + 1 AND FF:  GOSUB 1850
1442  N =  SGN (A AND 128): Z = 1 -  SGN (A):  GOTO 990
1450  REM  INX
1451  XR = (XR + 1) AND FF
1452  Z = 1 -  SGN (XR): N =  SGN (XR AND 128):  GOTO 990
1460  REM  INY
1461  YR = (YR + 1) AND FF
1462  Z = 1 -  SGN (YR): N =  SGN (YR AND 128):  GOTO 990
1470  REM  JMP
1471  GOSUB 1900: PC = AD:  GOTO 1000
1480  REM  JSR
1481  A = PC + 2: PH =  INT (A / HI): PL = A - PH * HI: SP(SP) = PH: SP
= SP - 1 AND FF: SP(SP) = PL: SP = SP - 1 AND FF
1482  PC =  PEEK (PC + 1) +  PEEK (PC + 2) * HI:  GOTO 1000
1490  REM  LDA
1491  GOSUB 1900: AC = OP:  GOTO 980
1500  REM  LDX
1501  GOSUB 1900: XR = OP:  GOTO 1452
1510  REM  LDY
1511  GOSUB 1900: YR = OP:  GOTO 1462
1520  REM  LSR
1521  IF AD( PEEK (PC)) <  > 4 THEN 1524
1522  AC = AC / 2
1523  C =  - (AC <  >  INT (AC)): AC = AC AND FF:  GOTO 980
1524  GOSUB 1900: A = OP / 2: C =  - (A <  >  INT (A)): A = A AND FF:
GOSUB 1850
1525  GOTO 1442
1530  REM  NOP
1531  GOTO 990
1540  REM  ORA
1541  GOSUB 1900: AC = AC OR OP:  GOTO 980
1550  REM  PHA
1551  SP(SP) = AC: SP = SP - 1 AND FF:  GOTO 990
```

```
1560  REM  PHP
1561  GOSUB 900: SP(SP) = SR: SP = SP - 1 AND FF:  GOTO 990
1570  REM  PLA
1571 SP = (SP + 1) AND FF: AC = SP(SP):  GOTO 980:  REM  FLAGS SETZEN
1580  REM  PLP
1581 SP = (SP + 1) AND FF: SR = SP(SP):  GOSUB 910:  GOTO 990
1590  REM  ROL
1591  IF AD( PEEK (PC)) = 4 THEN AC = AC * 2 + C:  GOTO 1523
1592  GOSUB 1900: A = OP * 2 + C: C =  - (A > FF)
1593 A = A AND FF:  GOSUB 1850
1594  GOSUB 1442
1600  REM  ROR
1601  IF AD( PEEK (PC)) = 4 THEN AC = AC / 2 + 128 * C:  GOTO 1523
1602  GOSUB 1900: A = OP / 2 + 128 * C: C =  - (A <  >  INT (A)):  GOTO
1593
1610  REM  RTI
1611 SP = SP + 1 AND FF: SR = SP(SP):  GOSUB 910:  GOTO 1621
1620  REM  RTS
1621 SP = SP + 1 AND FF: A = SP(SP): SP = SP + 1 AND FF: PC = A +
SP(SP) * HI:  GOTO 990
1630  IF D THEN 1635:  REM  SBC
1631  GOSUB 1900: V =  SGN (AC AND 128): AC = AC - OP - 1 + C: C =  -
(AC >  = 0)
1632 AC = AC AND FF: N =  SGN (AC AND 128): V = V AND 1 - N:  GOTO 980
1635  GOSUB 1900: AC =  VAL (H$(AC / 16) + H$(AC AND 15)): OP =  VAL
(H$(OP / 16) + H$(OP AND 15))
1636 AC = AC - OP + C - 1: C =  - (AC >  = 0):  IF AC < 0 THEN AC = AC
+ 100
1637 A$ =  MID$ ( STR$ (AC),2):  GOSUB 2390: AC = A:  GOTO 980
1640  REM  SEC
1641 C = 1:  GOTO 990
1650  REM  SED
1651 D = 1:  GOTO 990
1660  REM  SEI
1661 I = 1:  GOTO 990
1670  REM  STA
1671  GOSUB 1900: A = AC:  GOSUB 1850
1672  GOTO 990
1680  REM  STX
1681  GOSUB 1900: A = XR:  GOSUB 1850
1682  GOTO 990
1690  REM  STY
1691  GOSUB 1900: A = YR:  GOSUB 1850
1692  GOTO 990
1700  REM  TAX
1701 XR = AC:  GOTO 1452
1710  REM  TAY
1711 YR = AC:  GOTO 1462
1720  REM  TSX
1721 XR = SP:  GOTO 1452
1730  REM  TXA
1731 AC = XR:  GOTO 980
1740  REM  TXS
```

```
1741  SP = XR:  GOTO 990
1750   REM  TYA
1751  AC = YR:  GOTO 980
1800   REM  BRANCH-BEFEHLE
1810   IF FL = 0 THEN L = 1:  GOTO 990
1820   GOSUB 1985:  GOTO 1000
1850   REM  POKE
1870   IF AD < HI OR AD > HI + FF THEN 1880
1875  SP(AD - HI) = A:  RETURN
1880   IF ES THEN  POKE AD,A
1885   RETURN
1900   REM  OPERAND HOLEN
1910  A = AD( PEEK (PC))
1920   ON A GOSUB
1930,1935,1940,1945,1950,1955,1960,1965,1970,1975,1980,1985,1990
1925   IF AD < HI OR AD > HI + FF THEN  RETURN
1927  OP = SP(AD - HI):  RETURN
1930  AD = 0:  RETURN :  REM  IMPLIED
1935  AD = PC + 1: OP =  PEEK (AD): L = 1:  RETURN :  REM  #
1940  AD =  PEEK (PC + 1): OP =  PEEK (AD): L = 1:  RETURN : ZEROPAGE
1945  AD = 0:  RETURN :  REM  A
1950  AD =  PEEK (PC + 1) + HI *  PEEK (PC + 2): OP =  PEEK (AD): L = 2:
RETURN :  REM  ABSOLUT
1955  AD =  PEEK (PC + 1) + XR AND FF: OP =  PEEK (AD): L = 1:  RETURN
1960  AD =  PEEK (PC + 1) + YR AND FF: OP =  PEEK (AD): L = 1:  RETURN
1965  AD =  PEEK (PC + 1) + HI *  PEEK (PC + 2) + XR: OP =  PEEK (AD): L
= 2:  RETURN :  REM  ABSOLUT,X
1970  AD =  PEEK (PC + 1) + HI *  PEEK (PC + 2) + YR: OP =  PEEK (AD): L
= 2:  RETURN :  REM  ABSOLUT,Y
1975  AD =  PEEK (PC + 1) + HI *  PEEK ( PEEK (PC + 1) + 1): OP =  PEEK
(AD): L = 1:  RETURN
1980  AD =  PEEK (PC + 1) + XR AND FF: AD =  PEEK (AD) + HI *  PEEK (AD
+ 1): OP =  PEEK (AD): L = 1:  RETURN
1985  A =  PEEK (PC + 1): A = A + HI * (A > 127) + 2 + PC
1986  PC =  INT (A / HI) * HI + ((A + (A > SC) * UL) AND FF):  RETURN :
REM  RELATIV
1990  AD =  PEEK (PC + 1) + HI *  PEEK (PC + 2): AD =  PEEK (AD) + HI *
PEEK (AD + 1): OP =  PEEK (AD):  RETURN
2040   FOR P = S TO E:  PRINT " ";
2050  A = P:  GOSUB 2290:  REM  ADRESSE
2060   PRINT " ";: A =  PEEK (P):  GOSUB 2320:  PRINT " ";: J =  PEEK
(P): OP = AD(J)
2070   ON OP GOSUB
2350,2360,2360,2350,2370,2360,2360,2370,2370,2360,2360,2360,2370
2080   PRINT " ";MN$(J)" ";
2090   ON OP GOSUB
2110,2120,2130,2140,2150,2160,2170,2180,2190,2200,2210,2220,2240
2100   PRINT "< 8 SPACE>":  NEXT P
2105   IF P >  = UL THEN P = P - UL
2110   RETURN
2120   PRINT "#";:  GOSUB 2330: P = P + 1:  RETURN
2130   GOSUB 2330: P = P + 1:  RETURN
2140   PRINT " A";:  RETURN
```

```
2150  GOSUB 2260: P = P + 2:  RETURN
2160  GOSUB 2330: P = P + 1:  PRINT ",X";:  RETURN
2170  GOSUB 2330: P = P + 1:  PRINT ",Y";:  RETURN
2180  GOSUB 2260: P = P + 2:  PRINT ",X";:  RETURN
2190  GOSUB 2260: P = P + 2:  PRINT ",Y";:  RETURN
2200  PRINT "(";:  GOSUB 2330: P = P + 1:  PRINT "),Y";:  RETURN
2210  PRINT "(";:  GOSUB 2330: P = P + 1:  PRINT ",X)";:  RETURN
2220 A =  PEEK (P + 1): A = A + HI * (A > 127) + 2 + P
2230 A =  INT (A / HI) * HI + ((A + (A > SC) * UL) AND FF):  PRINT
"$";:  GOSUB 2290: P = P + 1:  RETURN
2240  PRINT "(";:  GOSUB 2260
2250  PRINT ")";: P = P + 2:  RETURN
2260  PRINT "$";
2270 A =  PEEK (P + 1) + HI *  PEEK (P + 2)
2280  REM  HEXADRESSE A
2290 HB =  INT (A / HI): A = A - HI * HB
2300  PRINT H$(HB / 16)H$(HB AND 15);
2310  REM  HEX-BYTE A
2320  PRINT H$(A / 16)H$(A AND 15);:  RETURN
2330  PRINT "$";
2340 A =  PEEK (P + 1):  GOTO 2320
2350  PRINT "< 5 SPACE>";:  RETURN
2360  GOSUB 2340:  PRINT "< 3 SPACE>";:  RETURN
2370  GOSUB 2340:  PRINT " ";: A =  PEEK (P + 2):  GOTO 2320
2380  IF  ASC (A$) = 42 THEN  END
2390 A = 0:  FOR J = 1 TO  LEN (A$): X =  ASC ( RIGHT$ (A$,J)) - 48: X
= X + (X > 9) * 7: A = A + X * (16 ^ (J - 1)):  NEXT :  RETURN
3000  PRINT :  PRINT "< 2 CRSR DOWN>":  PRINT "ADRESSE: < 3
SPACE>$****< 6 CRSR LEFT>";:  INPUT A$:  GOSUB 2380
3010  PRINT "<CRSR UP>",,: AD = A: OP =  PEEK (A):  GOSUB 1925: A = OP:
GOSUB 2320:  INPUT "< 4 CRSR LEFT>";A$:  GOSUB 2380
3020  GOSUB 1850:  PRINT "<CRSR UP>< 29 SPACE>":  IF AD = PC THEN 1000
3030  GOTO 200
3100  INPUT "ECHTSIMULATION< 3 SPACE>J< 3 CRSR LEFT>";ES$: ES = ES$ =
"J":  GOTO 200
10000  DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
10010  DATA "BRK",11,1,"ORA",35,11,"???",0,1
10020  DATA "???",0,1,"???",0,1,"ORA",35,3
10030  DATA "ASL",3,3,"???",0,1,"PHP",37,1
10040  DATA "ORA",35,2,"ASL",3,4,"???",0,1
10050  DATA "???",0,1,"ORA",35,5,"ASL",3,5
10060  DATA "???",0,1,"BPL",10,12,"ORA",35,10
10070  DATA "???",0,1,"???",0,1,"???",0,1
10080  DATA "ORA",35,6,"ASL",3,6,"???",0,1
10090  DATA "CLC",14,1,"ORA",35,9,"???",0,1
10100  DATA "???",0,1,"???",0,1,"ORA",35,8
10110  DATA "ASL",3,8,"???",0,1,"JSR",29,5
10120  DATA "AND",2,11,"???",0,1,"???",0,1
10130  DATA "BIT",7,3,"AND",2,3,"ROL",40,3
10140  DATA "???",0,1,"PLP",39,1,"AND",2,2
10150  DATA "ROL",40,4,"???",0,1,"BIT",7,5
10160  DATA "AND",2,5,"ROL",40,5,"???",0,1
10170  DATA "BMI",8,12,"AND",2,10,"???",0,1
```

```
10180  DATA "???",0,1,"???",0,1,"AND",2,6
10190  DATA "ROL",40,6,"???",0,1,"SEC",45,1
10200  DATA "AND",2,9,"???",0,1,"???",0,1
10210  DATA "???",0,1,"AND",2,8,"ROL",40,8
10220  DATA "???",0,1,"RTI",42,1,"EOR",24,11
10230  DATA "???",0,1,"???",0,1,"???",0,1
10240  DATA "EOR",24,3,"LSR",33,3,"???",0,1
10250  DATA "PHA",36,1,"EOR",24,2,"LSR",33,4
10260  DATA "???",0,1,"JMP",28,5,"EOR",24,5
10270  DATA "LSR",33,5,"???",0,1,"BVC",12,12
10280  DATA "EOR",24,10,"???",0,1,"???",0,1
10290  DATA "???",0,1,"EOR",24,6,"LSR",33,6
10300  DATA "???",0,1,"CLI",16,1,"EOR",24,9
10310  DATA "???",0,1,"???",0,1,"???",0,1
10320  DATA "EOR",24,8,"LSR",33,8,"???",0,1
10330  DATA "RTS",43,1,"ADC",1,11,"???",0,1
10340  DATA "???",0,1,"???",0,1,"ADC",1,3
10350  DATA "ROR",41,3,"???",0,1,"PLA",38,1
10360  DATA "ADC",1,2,"ROR",41,4,"???",0,1
10370  DATA "JMP",28,13,"ADC",1,5,"ROR",41,5
10380  DATA "???",0,1,"BVS",13,12,"ADC",1,10
10390  DATA "???",0,1,"???",0,1,"???",0,1
10400  DATA "ADC",1,6,"ROR",41,6,"???",0,1
10410  DATA "SEI",47,1,"ADC",1,9,"???",0,1
10420  DATA "???",0,1,"???",0,1,"ADC",1,8
10430  DATA "ROR",41,8,"???",0,1,"???",0,1
10440  DATA "STA",48,11,"???",0,1,"???",0,1
10450  DATA "STY",50,3,"STA",48,3,"STX",49,3
10460  DATA "???",0,1,"DEY",23,1,"???",0,1
10470  DATA "TXA",54,1,"???",0,1,"STY",50,5
10480  DATA "STA",48,5,"STX",49,5,"???",0,1
10490  DATA "BCC",4,12,"STA",48,10,"???",0,1
10500  DATA "???",0,1,"STY",50,6,"STA",48,6
10510  DATA "STX",49,7,"???",0,1,"TYA",56,1
10520  DATA "STA",48,9,"TXS",55,1,"???",0,1
10530  DATA "???",0,1,"STA",48,8,"???",0,1
10540  DATA "???",0,1,"LDY",32,2,"LDA",30,11
10550  DATA "LDX",31,2,"???",0,1,"LDY",32,3
10560  DATA "LDA",30,3,"LDX",31,3,"???",0,1
10570  DATA "TAY",52,1,"LDA",30,2,"TAX",51,1
10580  DATA "???",0,1,"LDY",32,5,"LDA",30,5
10590  DATA "LDX",31,5,"???",0,1,"BCS",5,12
10600  DATA "LDA",30,10,"???",0,1,"???",0,1
10610  DATA "LDY",32,6,"LDA",30,6,"LDX",31,7
10620  DATA "???",0,1,"CLV",17,1,"LDA",30,9
10630  DATA "TSX",53,1,"???",0,1,"LDY",32,8
10640  DATA "LDA",30,8,"LDX",31,9,"???",0,1
10650  DATA "CPY",20,2,"CMP",18,11,"???",0,1
10660  DATA "???",0,1,"CPY",20,3,"CMP",18,3
10670  DATA "DEC",21,3,"???",0,1,"INY",27,1
10680  DATA "CMP",18,2,"DEX",22,1,"???",0,1
10690  DATA "CPY",20,5,"CMP",18,5,"DEC",21,5
10700  DATA "???",0,1,"BNE",9,12,"CMP",18,10
```

```
10710 DATA "???",0,1,"???",0,1,"???",0,1
10720 DATA "CMP",18,6,"DEC",21,6,"???",0,1
10730 DATA "CLD",15,1,"CMP",18,9,"???",0,1
10740 DATA "???",0,1,"???",0,1,"CMP",18,8
10750 DATA "DEC",21,8,"???",0,1,"CPX",19,2
10760 DATA "SBC",44,11,"???",0,1,"???",0,1
10770 DATA "CPX",19,3,"SBC",44,3,"INC",25,3
10780 DATA "???",0,1,"INX",26,1,"SBC",44,2
10790 DATA "NOP",34,1,"???",0,1,"CPX",19,5
10800 DATA "SBC",44,5,"INC",25,5,"???",0,1
10810 DATA "BEQ",6,12,"SBC",44,10,"???",0,1
10820 DATA "???",0,1,"???",0,1,"SBC",44,6
10830 DATA "INC",25,6,"???",0,1,"SED",46,1
10840 DATA "SBC",44,9,"???",0,1,"???",0,1
10850 DATA "???",0,1,"SBC",44,8,"INC",25,8
10860 DATA "???",0,1
```

4.10 Ein Disassembler für den 6510

In diesem Unterkapitel finden Sie einen vollständigen Disassembler für den 6510-Prozessor. Zweck eines solchen Programms ist es, Maschinenprogramme, die im Speicher des Rechners stehen, in die symbolischen Bezeichnungen zurück zu übersetzen, die man auch bei der Eingabe von Machinenprogrammen benutzt. Aus der Byte-Folge $a9, $80 macht der Disassembler z.B. LDA #$80. Der Disassembler selbst wird einfach mit RUN gestartet und fragt dann nach der Start- und Endadresse des Bereichs, den er disassemblieren soll.

Noch kurz etwas zur Arbeitsweise des Disassemblers: Das Programm holt sich ein Byte aus dem Speicher und interpretiert dies als Befehlscode. Dieser Befehlscode dient als Index in einer Tabelle, aus der das Befehlswort, z.B. LDA, sowie die Adressierungsart des Befehls geholt werden. Aus der Länge weiß der Disassembler, wie lang der Befehl ist und in welcher Form der Operand anzugeben ist. Dies geschieht dann über Unterprogramme.

Hier nun das Listing des Disassemblers:

PROGRAMM:DISASSEMBLER

```
100 REM 6510 - DISASSEMBLER
110 DIM MN$(255),AD(255),H$(15)
```

```
120  FF = 255: HI = 256: UL = 2 ^ 16: SC = 2 ^ 15 - 1
130   PRINT "<CLR HOME>< 3 CRSR DOWN>< 8 CRSR RIGHT>6510 - DISASSEMBLER"
140   FOR I = 0 TO 15:  READ H$(I):  NEXT
150   FOR I = 0 TO 255:  READ MN$(I),AD(I):  NEXT
160   PRINT "<CRSR DOWN>STARTADRESSE: -< 3 SPACE>$****< 6 CRSR LEFT>";:
INPUT A$
170   GOSUB 540: S = A
180   PRINT "<CRSR DOWN>ENDADRESSE< 2 SPACE>: -< 3 SPACE>$****< 6 CRSR
LEFT>";:  INPUT A$:  PRINT
190   GOSUB 540: E = A
200   FOR P = S TO E
210  A = P:  GOSUB 450:  REM  ADRESSE
220   PRINT " ";: A =  PEEK (P):  GOSUB 480:  PRINT " ";: I =  PEEK (P):
OP = AD(I)
230   ON OP GOSUB 510,520,520,510,530,520,520,530,530,520,520,520,530
240   PRINT " ";MN$(I)" ";
250   ON OP GOSUB 270,280,290,300,310,320,330,340,350,360,370,380,400
260   NEXT P:  GOTO 160
270   PRINT :  RETURN
280   PRINT "#";:  GOSUB 490: P = P + 1:  PRINT :  RETURN
290   GOSUB 490: P = P + 1:  PRINT :  RETURN
300   PRINT " A":  RETURN
310   GOSUB 420: P = P + 2:  PRINT :  RETURN
320   GOSUB 490: P = P + 1:  PRINT ",X":  RETURN
330   GOSUB 490: P = P + 1:  PRINT ",Y":  RETURN
340   GOSUB 420: P = P + 2:  PRINT ",X":  RETURN
350   GOSUB 420: P = P + 2:  PRINT ",Y":  RETURN
360   PRINT "(";:  GOSUB 490: P = P + 1:  PRINT "),Y":  RETURN
370   PRINT "(";:  GOSUB 490: P = P + 1:  PRINT ",X)":  RETURN
380  T =  PEEK (P + 1): Q = T + HI * (T > 127) + 2 + P
390  A =  INT (Q / HI) * HI + ((Q + (Q > SC) * UL) AND FF):  PRINT "$";:
GOSUB 450: P = P + 1:  PRINT :  RETURN
400   PRINT "(";:  GOSUB 420
410   PRINT ")": P = P + 2:  RETURN
420   PRINT "$";
430  A =  PEEK (P + 1) + HI *  PEEK (P + 2)
440   REM  HEXADRESSE A
450  HB =  INT (A / HI): A = A - HI * HB
460   PRINT H$(HB / 16)H$(HB AND 15);
470   REM  HEX-BYTE A
480   PRINT H$(A / 16)H$(A AND 15);:  RETURN
490   PRINT "$";
500  A =  PEEK (P + 1):  GOTO 480
510   PRINT "< 5 SPACE>";:  RETURN
520   GOSUB 500:  PRINT "< 3 SPACE>";:  RETURN
530   GOSUB 500:  PRINT " ";: A =  PEEK (P + 2):  GOTO 480
540   IF  ASC (A$) = 42 THEN  END
550  A = 0:  FOR I = 1 TO 4: V =  ASC ( RIGHT$ (A$,I)) - 48: V = V + (V
> 9) * 7: A = A + V * (16 ^ (I - 1)):  NEXT :  RETURN
1000   DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
1010   DATA "BRK",1,"ORA",11,"???",1
1020   DATA "???",1,"???",1,"ORA",3
1030   DATA "ASL",3,"???",1,"PHP",1
```

```
1040  DATA "ORA",2,"ASL",4,"???",1
1050  DATA "???",1,"ORA",5,"ASL",5
1060  DATA "???",1,"BPL",12,"ORA",10
1070  DATA "???",1,"???",1,"???",1
1080  DATA "ORA",6,"ASL",6,"???",1
1090  DATA "CLC",1,"ORA",9,"???",1
1100  DATA "???",1,"???",1,"ORA",8
1110  DATA "ASL",8,"???",1,"JSR",5
1120  DATA "AND",11,"???",1,"???",1
1130  DATA "BIT",3,"AND",3,"ROL",3
1140  DATA "???",1,"PLP",1,"AND",2
1150  DATA "ROL",4,"???",1,"BIT",5
1160  DATA "AND",5,"ROL",5,"???",1
1170  DATA "BMI",12,"AND",10,"???",1
1180  DATA "???",1,"???",1,"AND",6
1190  DATA "ROL",6,"???",1,"SEC",1
1200  DATA "AND",9,"???",1,"???",1
1210  DATA "???",1,"AND",8,"ROL",8
1220  DATA "???",1,"RTI",1,"EOR",11
1230  DATA "???",1,"???",1,"???",1
1240  DATA "EOR",3,"LSR",3,"???",1
1250  DATA "PHA",1,"EOR",2,"LSR",4
1260  DATA "???",1,"JMP",5,"EOR",5
1270  DATA "LSR",5,"???",1,"BVC",12
1280  DATA "EOR",10,"???",1,"???",1
1290  DATA "???",1,"EOR",6,"LSR",6
1300  DATA "???",1,"CLI",1,"EOR",9
1310  DATA "???",1,"???",1,"???",1
1320  DATA "EOR",8,"LSR",8,"???",1
1330  DATA "RTS",1,"ADC",11,"???",1
1340  DATA "???",1,"???",1,"ADC",3
1350  DATA "ROR",3,"???",1,"PLA",1
1360  DATA "ADC",2,"ROR",4,"???",1
1370  DATA "JMP",13,"ADC",5,"ROR",5
1380  DATA "???",1,"BVS",12,"ADC",10
1390  DATA "???",1,"???",1,"???",1
1400  DATA "ADC",6,"ROR",6,"???",1
1410  DATA "SEI",1,"ADC",9,"???",1
1420  DATA "???",1,"???",1,"ADC",8
1430  DATA "ROR",8,"???",1,"???",1
1440  DATA "STA",11,"???",1,"???",1
1450  DATA "STY",3,"STA",3,"STX",3
1460  DATA "???",1,"DEY",1,"???",1
1470  DATA "TXA",1,"???",1,"STY",5
1480  DATA "STA",5,"STX",5,"???",1
1490  DATA "BCC",12,"STA",10,"???",1
1500  DATA "???",1,"STY",6,"STA",6
1510  DATA "STX",7,"???",1,"TYA",1
1520  DATA "STA",9,"TXS",1,"???",1
1530  DATA "???",1,"STA",8,"???",1
1540  DATA "???",1,"LDY",2,"LDA",11
1550  DATA "LDX",2,"???",1,"LDY",3
1560  DATA "LDA",3,"LDX",3,"???",1
```

```
1570  DATA "TAY",1,"LDA",2,"TAX",1
1580  DATA "???",1,"LDY",5,"LDA",5
1590  DATA "LDX",5,"???",1,"BCS",12
1600  DATA "LDA",10,"???",1,"???",1
1610  DATA "LDY",6,"LDA",6,"LDX",7
1620  DATA "???",1,"CLV",1,"LDA",9
1630  DATA "TSX",1,"???",1,"LDY",8
1640  DATA "LDA",8,"LDX",9,"???",1
1650  DATA "CPY",2,"CMP",11,"???",1
1660  DATA "???",1,"CPY",3,"CMP",3
1670  DATA "DEC",3,"???",1,"INY",1
1680  DATA "CMP",2,"DEX",1,"???",1
1690  DATA "CPY",5,"CMP",5,"DEC",5
1700  DATA "???",1,"BNE",12,"CMP",10
1710  DATA "???",1,"???",1,"???",1
1720  DATA "CMP",6,"DEC",6,"???",1
1730  DATA "CLD",1,"CMP",9,"???",1
1740  DATA "???",1,"???",1,"CMP",8
1750  DATA "DEC",8,"???",1,"CPX",2
1760  DATA "SBC",11,"???",1,"???",1
1770  DATA "CPX",3,"SBC",3,"INC",3
1780  DATA "???",1,"INX",1,"SBC",2
1790  DATA "NOP",1,"???",1,"CPX",5
1800  DATA "SBC",5,"INC",5,"???",1
1810  DATA "BEQ",12,"SBC",10,"???",1
1820  DATA "???",1,"???",1,"SBC",6
1830  DATA "INC",6,"???",1,"SED",1
1840  DATA "SBC",9,"???",1,"???",1
1850  DATA "???",1,"SBC",8,"INC",8
1860  DATA "???",1
```

4.11 Die 64er-Maus 1351

Die Maus ist ein sehr leicht zu bedienendes Eingabeinstrument. Sie belegt die gleichen Anschlüsse wie auch der Joystick.

Doch wie funktioniert eigentlich die Maus? Die Bewegung der in der Maus befindlichen Kugel wird auf zwei Rollen übertragen. Die eine Rolle übernimmt die Bewegung in X- und die andere in Y-Richtung. An jeder der Rollen ist ein Kranz mit Löchern befestigt. Die Löcher lassen einen Lichtstrahl, der auf einen Fotowiderstand fällt, ungehindert durch. Durch Drehen des Kranzes entsteht ein ständiger Wechsel zwischen hell und dunkel. Die Bewegung der Maus wird somit nicht wie beim Joystick durch das Vorhanden- und Nichtvorhandensein des Signals erkannt.

Das folgende Maschinenprogramm ermöglicht es, die Commodore-Maus 1351 in Port 1, die speziell für den C64 entwickelt wurde, abzufragen. Es liegt in $8000, von wo es nach $C000 kopiert wird. In $C000 liegt das Interrupt-Programm, das die Maus steuert. Das Mausprogramm erkennt, welches Zeichen sich unter dem Mauszeiger befindet. Außerdem lassen sich die X- und Y-Koordinaten in den Adressen $037D und $037E abfragen.

Nachdem das Maschinenprogramm in den Speicher geladen wurde, kann es mit SYS 32832 gestartet werden. Im BASIC-Loader ist dies bereits im Programm eingebunden und braucht nur mit RUN gestartet zu werden. Nun folgt das Maschinen-Listing und der BASIC-Loader:

```
Sprite-Daten

8000 F8 00 00 90 00 00 B8 00
8008 00 DC 00 00 8E 00 00 07
8010 00 00 02 00 00 00 00 00
8018 00 00 00 00 00 00 00 00
8020 00 00 00 00 00 00 00 00
8028 00 00 00 00 00 00 00 00
8030 00 00 00 00 00 00 00 00
8038 00 00 00 00 00 00 00 00


Programmanfang

8040 LDY #$00    Zähler auf Null
8042 LDA $8000,Y Sprite-Daten
8045 STA $0E00,Y nach Puffer 1 kopieren
8048 INY         Zähler erhöhen
8049 CPY #$40    wenn fertig,
804B BNE $8042   dann weiter
804D LDA #$01    Sprite 1
804F STA $D015   einschalten
8052 STA $D027   und Farbe setzen
8055 LDA #$64    Sprite
8057 STA $D000   X-Richtung
805A STA $D001   Y-Richtung
805D LDA #$00    Überlauf-Register
805F STA $D010   setzen
8062 LDA #$38    Zeiger auf
8064 STA $07F8   Puffer 1
8067 LDY #$00    Zähler auf Null
8069 LDA $807C,Y Interrupt-Routine nach
806C STA $C000,Y $C000 kopieren
806F INY         Zähler erhöhen
```

```
8070 CPY #$01    wenn fertig,
8072 BNE $8069   dann weiter
8074 LDA #$C0    Wert
8076 STA $0A04   zwischenspeichern
8079 JMP $C000   Sprung zur Interrupt-Routine
807C NOP
807D NOP
807E NOP
807F NOP
8080 NOP
8081 NOP
8082 NOP
8083 NOP
8084 NOP
8085 NOP
8086 NOP
8087 NOP
8088 NOP
8089 NOP
808A NOP
808B NOP
808C NOP
808D NOP
808E NOP
808F SEI         Interrupt sperren
8090 LDA #$21    IRQ auf
8092 STA $0314   $C021
8095 LDA #$C0    setzen
8097 STA $0315   und
809A PLP         Register holen
809B CLI         Interrupt freigeben
809C RTS         Rücksprung
809D NOP         keine Operation
809E LDA $D419   Mauskoordinate holen
80A1 LDY $C0D1
80A4 JSR $C058
80A7 STY $C0D1
80AA CLC         Carry für Addition löschen
80AB ADC $D000   zur alten Position addieren
80AE STA $D000   und speichern
80B1 TXA         X-Reg. nach Akku
80B2 ADC #$00    auf
80B4 AND #$01    Überlauf
80B6 EOR $D010   prüfen
80B9 STA $D010   und speichern
80BC LDA $D41A   Mauskoordinaten holen
80BF LDY $C0D2
80C2 JSR $C058
80C5 STY $C0D2
80C8 SEC         Carry für Addition
80C9 EOR #$FF    Bits umdrehen
80CB ADC $D001   Zur alten Position addieren
80CE STA $D001   und speichern
```

```
80D1 JMP $C083   Rücksprung
80D4 STY $C0D4   Werte
80D7 STA $C0D3   zwischenspeichern
80DA LDX #$00    X-Reg. löschen
80DC SEC         Carry für Subtraktion löschen
80DD SBC $C0D4   subtrahieren
80E0 AND #$7F    Bit 8 löschen
80E2 CMP #$40    wenn Werte gleich,
80E4 BCS $80ED   dann verzweige
80E6 LSR         mal 2
80E7 BEQ $80FB   verzweige, wenn kein überlauf
80E9 LDY $C0D3   alten Wert laden
80EC RTS         Rücksprung
80ED ORA #$C0    Bits setzen
80EF CMP #$FF    und vergleichen
80F1 BEQ $80FB   verzweige, wenn kein Überlauf
80F3 SEC         Carry für Rechenoperation
80F4 ROR         Bits verschieben
80F5 LDX #$FF    alte Werte
80F7 LDY $C0D3   holen
80FA RTS         Rücksprung
80FB LDA #$00    Akku löschen
80FD RTS         Rücksprung
80FE BRK
80FF LDA $D000   X-Position laden
8102 SBC #$18    und subtrahieren
8104 LDX #$FF    Wert laden
8106 INX         und erhöhen
8107 SBC #$08    so lange subtrahieren
8109 BCS $8106   bis Überlauf
810B TXA         X-Reg. wieder nach Akku
810C SBC #$01    subtrahieren
810E STX $03FD   und speichern
8111 SEC         Carry setzen
8112 LDA $D001   Y-Position laden
8115 SBC #$32    und subtrahieren
8117 LDX #$FF    Wert laden
8119 INX         und erhöhen
811A SBC #$08    so lange subtrahieren
811C BCS $8119   bis Überlauf
811E TXA         X-Reg. nach Akku
811F SBC #$01    subtrahieren
8121 STX $03FE   und speichern
8124 LDA $D010   Prüfen ob
8127 CMP #$01    Überlauf
8129 BNE $8148   verzweige, wenn ja
812B LDA $03FD   Puffer
812E CMP #$1D    richtig ?
8130 BEQ $8148   verzweige, wenn nein
8132 LDA $03FD   Puffer
8135 CMP #$1E    richtig ?
8137 BEQ $8148   verzweige, wenn nein
8139 LDA $03FD   Puffer
```

```
813C CMP #$1F     richtig ?
813E BEQ $8148    verzweige, wenn nein
8140 LDA $03FD    Puffer laden
8143 ADC #$20     richtig stellen
8145 STA $03FD    und speichern
8148 JMP $EA31    Sprung zur Interrupt-Routine
```

Und hier der BASIC-Loader:

```
100 FORI=32768TO33100:READA:POKEI,A:NEXT
105 SYS 32832
110 DATA248,0,0,144,0,0,184,0,0,220,0,0,142,0,0,7,0,0
120 DATA2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
130 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
140 DATA0,0,0,0,0,0,0,0,0,0,160,0,185,0,128,153,0,14
150 DATA200,192,64,208,245,169,1,141, 21,208,141,
39,208,169,100,141,0,208
160 DATA141,1,208,169,0,141,16,208,169,56,141,248,7,160,0,185,124,128
170 DATA153,0,192,200,192,209,208,245,169,192,141,4,10,76,0,192,234,234
180 DATA234,234,234,234,234,234,234,234,234,234,234,234,234,234,234,234,
234,120
190 DATA169,33,141,20,3,169,192,141,21,3,40,88,96,234,173,25,212,172
200 DATA209,192,32,88,192,140,209,192,24,109,0,208,141,0,208,138,105,0
210 DATA41,1,77,16,208,141,16,208,173,26,212,172,210,192,32, 88,192,140
220 DATA210,192,56,73,255,109,1,208,141,1,208,76,131,192,140,212,192,141
230 DATA211,192,162,0,56,237,212,192,41,127,201,64,176,7,74,240,18,172
240 DATA211,192,96,9,192,201,255,240,8,56,106,162,255,172,211,192,96,169
250 DATA0,96,0,173,0,208,233,24,162,255,232,233,8,176,251,138,233,1
260 DATA142,253,3,56,173,1,208,233,50,162,255,232,233,8,176,251,138,233
270 DATA1,142,254,3,173,16,208,201,1,208,29,173,253,3,201,29,240,22
280 DATA173,253,3,201,30,240,15,173,253,3,201,31,240,8,173,253,3,105
290 DATA32,141,253,3,76,49,234,0,0
```

5. Die Floppy VC 1541

In praktisch jedem Ihrer Programme haben Sie irgendwelche Daten zu verwalten. Selbst bei nur kleinen Datenmengen lohnt es sich dabei, diese Daten in separaten Dateien auf einer Diskette auszulagern. Die Verwaltung der Daten gestaltet sich dadurch aber wesentlich einfacher und vor allem flexibler als etwa bei der Arbeit mit DATA-Zeilen.

Und auch wenn sie "nur" Ihre Programme auf Diskette speichern möchten, sollten Sie ein wenig darüber Bescheid wissen, wie die Floppy arbeitet und wie eine Diskette organisiert ist.

In diesem Kapitel möchte ich Sie daher umfassend über alle Aspekte der Floppy-Programmierung informieren. Sie lernen, wie man der Floppy Befehle übermittelt, beispielsweise, um eine Datei zu löschen oder um ihr einen anderen Namen zu geben, Sie erfahren die Unterschiede zwischen der sequentiellen und relativen Datenspeicherung und vieles mehr.

Anmerkung: Die Floppy 1541, über die es in diesem Kapitel gehen soll, ist das "Standardlaufwerk" für den Commodore 64. In der Zwischenzeit gibt es von Commodore aber auch einige andere Modelle (1541-II, 1570/71, 1581), die ebenfalls am Commodore 64 betrieben werden können. Diese Geräte sind - zumindest nach Aussagen von Commodore - völlig "befehlskompatibel" zur 1541. D.h., sie sind zwar intern anders aufgebaut, "verstehen" aber sämtliche Befehle, die man der 1541 geben kann, selbst die Spezialbefehle am Ende dieses Kapitels. Wenn Sie also zufällig eines dieser Modelle besitzen sollten, so ist dieses Kapitel für Sie ebenfalls von Nutzen.

5.1 Die Ansteuerung der Floppy

Die Floppy 1541 zählt zu den sogenannten "intelligenten" Laufwerken. Sie verfügt über ein eigenständiges Betriebssystem, das die Arbeit der Floppy unabhängig vom Commodore 64 steuert.

Möchte man daher eine bestimmte Aktion bewirken, beispielsweise eine Datei löschen, so muß man nur einen entsprechenden Befehl senden. Den Rest erledigt die Floppy selbst. Am Ende des Vorgangs bekommt man gegebenenfalls eine Rückmeldung der Floppy, beispielsweise die Anzahl der gelöschten Dateien.

Wie sieht nun die Befehlsübermittlung an die Floppy aus?

Befehlsübermittlung in BASIC

Zunächst muß mit Hilfe der OPEN-Anweisung der sogenannte "Befehlskanal" zur Floppy geöffnet werden:

```
OPEN LF,8,15
```

Für die logische File-Nummer LF dürfen Sie Werte zwischen 1 und 127 verwenden. Die logische File-Nummer signalisiert dem Commodore 64 später, auf welche Datei Sie jeweils zugreifen wollen. Die 8 ist die Geräteadresse der Floppy. Bei der 15 dahinter handelt es sich um die sogenannte Sekundäradresse, die der Floppy in diesem Fall mitteilt, daß Sie einen Befehl über den Befehlskanal senden wollen. Nach dem Öffnen des Befehlskanals können Sie jederzeit mit

```
PRINT#LF,"BEFEHL"
```

einen Befehl an die Floppy senden. Zum Schluß müssen Sie den Befehlskanal mit

```
CLOSE LF
```

wieder schließen. Das ganze gliedert sich also in drei Phasen:

- Befehlskanal öffnen: `OPEN 15,8,15`
- Befehl senden: `PRINT#15,"BEFEHL"`
- Befehlskanal schließen: `CLOSE 15`

Falls Sie nur einen einzigen Befehl senden wollen, läßt sich das ganze auch etwas abkürzen:

```
OPEN 15,8,15,"BEFEHL":CLOSE 15
```

Im Prinzip können Sie den Befehlskanal während der ganzen Zeit, in der Sie mit der Floppy arbeiten, geöffnet halten. Wenn Sie im Direktmodus arbeiten, müssen Sie allerdings vorsichtig sein. Durch das Einladen eines BASIC-Programms in den Rechnerspeicher wird der Befehlskanal nämlich automatisch geschlossen! Ein späteres PRINT#15 hätte dann einen FILE NOT OPEN ERROR zur Folge.

Im Direktmodus ist es daher am besten, wenn man die abgekürzte Variante der Befehlsübermittlung verwendet. In den meisten Fällen sendet man ohnehin jeweils nur einen Befehl. Bevor wir die ersten Befehle betrachten, möchte ich Ihnen erst noch zeigen, wie die Floppy in Assembler angesteuert wird.

Befehlsübermittlung in Assembler

Die logische Abfolge der einzelnen Schritte ist in Assembler genau dieselbe wie in BASIC. Zuerst wird der Befehlskanal geöffnet, dann der Befehl gesendet und schließlich der Befehlskanal wieder geschlossen. Da für alle diese Zwecke entsprechende Routinen des Betriebssystems zur Verfügung stehen, wird das ganze zwar etwas länger, aber nicht viel komplizierter:

```
100 ;*** routine: befehl senden ***
110 ;
120 .ba 49152   ;* startadresse *
130 ;
140         lda #$0f        ;logische file-nr. 15
150         ldx #$08        ;geraeteadresse
160         ldy #$0f        ;sekundaeradr. 15
170         jsr $ffba       ;parameter setzen
180 ;
190         lda #$0b        ;befehlslaenge!
200         ldx #<(befehl) ;zeiger auf
210         ldy #>(befehl) ;befehlstext
220         jsr $ffbd       ;parameter setzen
230 ;
240         jsr $ffc0       ;open-routine aufrufen
250 ;
260         lda #$0f        ;logische file-nr. 15
270         jsr $ffc3       ;close-routine aufrufen
280 ;
290 befehl  .tx "befehlstext"
```

Der Einfachheit halber habe ich hier die verkürzte Form der Befehlsübermittlung verwendet. Der Befehlstext wird direkt mit OPEN gesendet und anschließend der Befehlskanal wieder geschlossen.

Formatieren von Disketten

Wie Sie bereits wissen, muß eine Diskette vor der ersten Benutzung formatiert werden. Dazu hält die Floppy den Befehl NEW parat:

```
"NEW:DISKETTENNAME,ID"
```

oder abgekürzt:

```
"N:DISKETTENNAME,ID"
```

Hinweis: Ich werde im folgenden immer die abgekürzte Form der Befehle verwenden, da man sich damit einiges an Tipparbeit erspart und die Befehle trotzdem noch gut lesbar sind.

Der Diskettenname darf bis zu 16 Zeichen lang sein. Anhand der dahinter angegebenen ID erkennt die Floppy einen Diskettenwechsel.

Es ist daher sehr wichtig, daß jede Ihrer Disketten eine andere ID besitzt. Die ID darf aus zwei beliebigen Zeichen oder Ziffern bestehen.

Am besten wird es sein, wenn Sie sich Ihre Disketten einfach von "01" bis "99" durchnumerieren. In BASIC sieht das Formatieren so aus:

```
100 rem *** routine: diskette formatieren ***
110 open 15,8,15
120 print#15,"n:arbeitsdisk,01"
130 close 15
```

Und das ganze in Assembler:

```
100 ;*** routine: diskette formatieren ***
110 ;
120 .ba 49152   ;* startadresse *
130 ;
140         lda #$0f       ;logische file-nr. 15
150         ldx #$08       ;geraeteadresse
160         ldy #$0f       ;sekundaeradr. 15
170         jsr $ffba      ;parameter setzen
180 ;
190         lda #$10       ;befehlslaenge!
200         ldx #<(befehl) ;zeiger auf
210         ldy #>(befehl) ;befehlstext
220         jsr $ffbd      ;parameter setzen
230 ;
240         jsr $ffc0      ;open-routine aufrufen
250 ;
260         lda #$0f       ;logische file-nr. 15
270         jsr $ffc3      ;close-routine aufrufen
280 ;
290 befehl  .tx "n:arbeitsdisk,01"
```

Der gesamte Formatiervorgang dauert etwa 90 Sekunden. Das laute Rattern ganz am Anfang sollte Sie nicht beunruhigen. Dabei wird nur der Schreib-/Lesekopf der Floppy justiert, um eine einwandfreie Formatierung der Diskette zu gewährleisten. Häufig hat man ältere Disketten, deren Inhalt man nicht mehr benötigt und die man deshalb für andere Zwecke verwenden möchte. Dazu muß man natürlich zunächst die alten Daten auf der Diskette löschen. Am einfachsten geht das, indem Sie die Diskette neu formatieren.

Dabei dürfen Sie die ID der Diskette auch weglassen (die Diskette behält dann die alte ID), wodurch sich der Formatiervorgang auf wenige Sekunden verkürzt. Beim Formatieren ohne ID wird nämlich nur das Inhaltsverzeichnis der Diskette gelöscht (damit sind dann automatisch auch sämtliche Daten gelöscht). Ein Beispiel:

```
110 open 15,8,15
120 print#15,"n:neue disk"
130 close 15
```

gibt der eingelegten Diskette den Namen "NEUE DISK".

Vorsicht: Beim Neuformatieren älterer Disketten sollte man sich grundsätzlich sehr sorgfältig davon überzeugen, daß man die gespeicherten Daten auch wirklich nicht mehr benötigt!

Fehlermeldungen der Floppy auslesen

Wie auch der Commodore 64 Sie gelegentlich durch Fehlermeldungen, wie etwa SYNTAX ERROR, darauf aufmerksam macht, daß Sie irgend etwas falsch gemacht bzw. falsch eingegeben haben, so reagiert auch die Floppy auf Fehler mit einer entsprechenden Meldung.

Daß ein Fehler aufgetreten ist, erkennen Sie am hektischen Blinken der roten LED an der Frontseite der Floppy. Wie kommt man nun aber an die Fehlermeldung heran?

Dazu müssen wir wieder den Befehlskanal öffnen und anschließend die Fehlermeldung einlesen:

```
100 rem *** routine: fehlermeldung auslesen ***
110 open 15,8,15
120 input#15,nr,ft$,tr,sk
(125 print nr;ft$;tr;sk)
130 close 15
```

Jede Fehlermeldung gliedert sich in vier Teile, die in die vier Variablen NR, FT$, TR und SK eingelesen werden.

NR: NR enthält die Nummer des aufgetretenen Fehlers.

FT$: FT$ enthält den Fehlertext.

TR,SK: Bei einigen Fehlermeldungen erhalten sie in TR und SK die Nummer der Spur und des Sektors des Datenblocks, in dem der Fehler auftrat. Ansonsten enhalten TR und SK den Wert Null.

Liegt im Augenblick des Auslesens kein Fehler vor, sendet die Floppy die Meldung

```
00 OK 00 00
```

Eine vollständige Liste aller Fehlermeldungen samt Beschreibung der Ursachen und möglicher Abhilfen finden Sie übrigens im Anhang. Ich werde aber auch in diesem Kapitel an den entsprechenden Stellen auf die Fehlermeldungen kurz eingehen. Die Fehlermeldung, die Sie wohl am häufigsten bekommen werden, dürfte

```
34 SYNTAX ERROR 00 00
```

sein. In diesem Fall haben Sie den betreffenden Befehl falsch geschrieben (vielleicht haben Sie einfach nur den Doppelpunkt oder ein Komma vergessen). Korrigieren Sie dann bitte den Befehl und versuchen es noch einmal. Die Fehlermeldung

```
62 FILE NOT FOUND 00 00
```

tritt auf, wenn man versucht, auf eine auf der eingelegten Diskette nicht vorhandene Datei zuzugreifen. Hier liegt der Fehler oft in einem falsch geschriebenen Dateinamen. In Assembler möchte ich mich auf's Einlesen der Fehlernummer beschränken:

```
100 ;*** routine: fehlernummer auslesen ***
110 ;
120 .ba 49152    ;* startadresse *
130 ;
140          lda #$0f       ;logische file-nr. 15
150          ldx #$08       ;geraeteadresse
160          ldy #$0f       ;sekundaeradr. 15
170          jsr $ffba      ;parameter setzen
180 ;
190          jsr $ffc0      ;open-routine aufrufen
200 ;
210          ldx #$0f       ;log. file-nr.
220          jsr $ffc6      ;eingabe aus logischer datei
230 ;
240          jsr $ffcf      ;1.ziffer holen
250          sta 251        ;und ablegen
260          jsr $ffcf      ;2.ziffer holen
270          sta 252        ;und ablegen
280 ;
290          jsr $ffcc      ;auf standardeingabe
                            ;zurückschalten
300          lda #$0f       ;logische file-nr. 15
310          jsr $ffc3      ;close-routine aufrufen
```

Die beiden Ziffern der Fehlernummer werden in den Speicherzellen 251 und 252 zwischengespeichert. Natürlich können Sie auch die gesamte Fehlermeldung mit JSR $FFCF einlesen und in einem Puffer ablegen. Das Ende der Meldung erkennen Sie an dem Code $OD ('Return'). Zu beachten ist, daß die Floppy die Meldung als String sendet. Lautet die Meldung also

```
00 OK 00 00
```

so stehen in 251 und 252 keine zwei Nullen, sondern der ASCII-Code 48 für Null!

Das Inhaltsverzeichnis einer Diskette

Auf jeder Diskette legt die Floppy ein Verzeichnis an, das sogenannte "Directory", in dem alle auf der Diskette gespeicherten Dateien vermerkt werden.

Das Directory läßt sich wie ein Programm in den Rechnerspeicher laden. Es hat den Namen "$" (das Dollarzeichen):

```
LOAD "$",8
```

Aber Vorsicht: Durch das Einladen des Directories geht ein evtl. im Rechnerspeicher vorhandenes BASIC-Programm verloren! Dieser Umstand ist sehr ärgerlich. Hat man zum Beispiel ein BASIC-Programm geschrieben und möchte dann vor dem Abspeichern nachsehen, ob die richtige Diskette eingelegt oder ob noch genügend Platz auf der Diskette vorhanden ist, so geht das nur, indem man das BASIC-Programm zerstört.

Aus diesem Grund habe ich eine kleine Maschinensprache-Routine geschrieben, die Ihnen das Directory ohne Programmverlust am Bildschirm ausgibt. Die Routine wird einfach mit

```
SYS 50002
```

aufgerufen. Durch Drücken der <Stop>-Taste können Sie die Ausgabe jederzeit abbrechen. Im Anschluß an das folgende Assembler-Listing finden Sie, wie gewohnt, einen BASIC-Lader zur Routine.

```
100 ;**********************************************************
105 ;*                                                        *
110 ;* programm: dir                                          *
120 ;* listet das directory einer diskette ohne               *
130 ;* programmverlust am bildschirm auf                      *
140 ;*                                                        *
150 ;**********************************************************
160 ;*                                                        *
170 ;* aufruf: sys 50002                                      *
190 ;*                                                        *
195 ;**********************************************************
200 ;
205 ;
210 .ba 50002    ;*** startadresse ***
220 ;
230 ;
240 ;*** labels **********************************************
250 .gl zw  =   2   ;zwischenspeicher
260 .gl zp1 = 251
265 ;
270 ;
275 ;*** initialisierung **************************************
280           lda #1          ;file-namen-laenge
285           ldx #<(dirname);file-namen-position
290           ldy #>(dirname)
300           jsr $ffbd       ;parameter setzen
310           lda #8          ;floppy-geraeteadresse
320           sta $ba
330           lda #$60        ;secundaeradresse
340           sta $b9
350           jsr $f3d5       ;file-namen senden
360           lda $ba         ;floppy auf senden schalten
370           jsr $ffb4       ;talk senden
380           lda $b9
390           jsr $ff96       ;sec.adresse senden
400           lda #0
410           sta $90         ;status-flag loeschen
420           jsr $aad7       ;return ausgeben
430 ;
440 ;*** directory holen und ausgeben ***********************
450           ldy #5          ;erste 5 zeichen ueberlesen
460 dr1       sty zw          ;wert zwischenspeichern
470           jsr $ffa5       ;ein byte einlesen
480           sta zp1         ;blockanzahl (low)
490           ldy zw          ;zaehler
500           dey
510           bne dr1         ;<>0, dann naechstes byte
520           jsr $ffa5       ;ein byte holen
530           ldy $90         ;file-ende erreicht?
540           bne dr4         ;ja, dann fertig
550           ldx zp1         ;blockanzahl (low)
560           jsr $bdcd       ;blockanzahl ausgeben
570           lda #" "        ;leerstelle ausgeben
```

```
580             jsr $ffd2
590 dr2         jsr $ffa5       ;ein byte holen
600             ldx $90         ;status testen
610             bne dr4         ;dir zu ende
620             tax
630             beq dr3         ;gleich 0, file-eintragende
640             jsr $ffd2       ;aktuelles zeichen ausgeben
650             jmp dr2         ;naechstes byte bearbeiten
660 dr3         lda #13         ;return ausgeben
670             jsr $ffd2
680             jsr $ffe1       ;stop-taste testen
690             beq dr4         ;gedrueckt, dann abbrechen
700             ldy #3
710             jmp dr1         ;naechste 3 zeichen ueberlesen
720 dr4         jsr $ffab       ;untalk senden
730             jmp $f642       ;file schliessen
740 ;
750 ;*** file-name fuer directory ***
760 dirname     .tx "$"
```

Und der BASIC-Lader:

```
100 rem *****************************************************
105 rem *                                                   *
110 rem * programm: dir                                     *
120 rem * programmlaenge: 106 bytes                         *
130 rem * listet das directory einer diskette ohne          *
140 rem * programmverlust am bildschirm auf                 *
150 rem *                                                   *
200 rem *****************************************************
205 rem *                                                   *
210 rem * aufruf: sys 50002                                 *
265 rem *                                                   *
270 rem *****************************************************
280 :
290 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=50002:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 169,1,162,187,160,195,32,189,255,169,1519
1010 data 8,133,186,169,96,133,185,32,213,243,1398
```

```
1020 data 165,186,32,180,255,165,185,32,150,255,1605
1030 data 169,0,133,144,32,215,170,160,5,132,1160
1040 data 2,32,165,255,133,251,164,2,136,208,1348
1050 data 244,32,165,255,164,144,208,41,166,251,1670
1060 data 32,205,189,169,32,32,210,255,32,165,1321
1070 data 255,166,144,208,24,170,240,6,32,210,1455
1080 data 255,76,150,195,169,13,32,210,255,32,1387
1090 data 225,255,240,5,160,3,76,121,195,32,1312
1100 data 171,255,76,66,246,36,0,0,0,0,850
2000 data -1:rem endmarkierung
```

Sehen wir uns einmal ein typisches Directory irgendeiner beliebigen Diskette an:

```
0    "datendisk       "  ad 2a
25   "dateiverwaltung"   prg
10   "index"             seq
70   "adressen"          rel
.....
473 blocks free.
```

In der revers geschriebenen Kopfzeile steht der Name der Diskette (DATENDISK) sowie ihre ID (AD). Die Kennung "2A" wird von der Floppy beim Formatieren automatisch erzeugt. An dieser Kennung erkennt die Floppy, daß die Diskette von ihr (bzw. einer anderen Commodore-Floppy) formatiert wurde.

Unterhalb der Kopfzeile sehen Sie die einzelnen Dateien. Die Namen in der Mitte dürften Ihnen klar sein. Wir haben also drei Dateien mit den Namen "DATEIVERWALTUNG", "INDEX" und "ADRESSEN".

Doch was bedeuten die Zahlen am Anfang und die dreistelligen Kürzel am Ende der Zeilen?

Die Zahlen geben die Größe der einzelnen Dateien an. Allerdings nicht in Byte, sondern in der Anzahl der Datenblöcke, die Sie auf der Diskette belegen. Dazu muß man wissen, daß eine Diskette (beim Formatieren) in 256 Byte große Blöcke unterteilt wird. Diese Blöcke, auch Sektoren genannt, wiederum sind zu konzentrischen Spuren (ähnlich den Rillen einer Schallplatte) zusammengefaßt.

Wie wir später noch sehen werden, läßt sich jeder dieser Datenblöcke durch Angabe seiner Spur- und Sektornummer auch direkt ansteuern. Man spricht dann von einem Direktzugriff auf die Diskette. Im Augenblick ist aber nur wichtig, daß Sie wissen, daß jede Datei in einzelnen Datenblöcken gespeichert wird. Die Blöcke sind dazu über einen Zeiger miteinander verkettet, d.h., jeder Datenblock enthält (ganz am Anfang) die Spur- und die Sektornummer des nächsten Blocks.

Aus diesem Grund passen in einen Block jeweils nur 254 Daten-Bytes.

Mit diesen Informationen läßt sich die tatsächliche Größe jeder Datei leicht berechnen:

```
Blockanzahl*254
```

Die Datei "DATEIVERWALTUNG" umfaßt also 25*254=6350 Bytes. Die Kürzel hinter den Dateinamen geben an, um welchen Dateityp es sich handelt:

PRG	steht für Programmdateien (BASIC- oder Maschinenprogramme, die mit SAVE gespeichert wurden).
SEQ	steht für sequentielle Dateien, in denen irgendwelche Daten gespeichert sind. Mehr im nächsten Abschnitt.
REL	steht für relative Dateien, in denen ebenfalls Daten abgelegt werden. Mehr im übernächsten Abschnitt.

In der letzten Zeile jedes Directories steht schließlich noch, wie viele Datenblocks frei, also noch nicht mit Daten belegt sind.

Bei unserer Beispieldiskette sind das im Augenblick 473 Blocks. Die Gesamtanzahl der Blöcke beträgt bei jeder Diskette 664. Auf einer Diskette lassen sich also maximal 664*254=168.656 Bytes speichern.

Um sich einen Überblick über seine Disketten- bzw. Programmsammlung zu verschaffen, ist es am einfachsten, wenn man sich die Directories der Disketten ausdruckt.

Das geht sehr leicht:

```
LOAD "$",8
OPEN 4,4
CMD 4
LIST
PRINT#4
CLOSE4
```

Disketten initialisieren

Vielleicht haben Sie sich in der Zwischenzeit schon gefragt, woher die Floppy eigentlich weiß, welcher Datenblock einer Diskette bereits mit Daten belegt ist und welcher noch nicht. Nun, um hier nicht die Übersicht zu verlieren, legt die Floppy ergänzend zum Directory für jede Diskette eine Blockbelegungstabelle (engl. Block Availability Map, kurz BAM) an.

Bei jedem Diskettenwechsel wird diese BAM von der Diskette in einen internen Speicherbereich der Floppy geholt. Einen Diskettenwechsel kann die Floppy aber nur erkennen, wenn die beiden Disketten verschiedene ID's besitzen. Ist das nicht der Fall, arbeitet die Floppy unter Umständen mit einer falschen BAM, was natürlich fatale Folgen haben kann.

Wenn Sie beispielsweise eine Diskette im Laufwerk hatten, die fast leer war, und anschließend eine nahezu volle Diskette (mit derselben ID) einlegen, um auf ihr ein Programm oder Daten abzuspeichern, dann sucht sich die Floppy die freien Blöcke aus der BAM der leeren Diskette und überschreibt dadurch höchstwahrscheinlich bereits auf der eingelegten Diskette gespeicherte Daten. Aus diesem Grund gibt es die Möglichkeit, die BAM "von Hand" in den Floppy-Speicher zu holen. Diesen Vorgang bezeichnet man auch als Initialisieren der Diskette. Der Befehl dafür lautet:

```
INITIALIZE
```

oder abgekürzt:

```
I
```

In BASIC sieht das so aus:

```
100 rem *** routine: diskette initialisieren ***
110 open 15,8,15
120 print#15,"i"
130 close 15
```

Und das ganze in Assembler:

```
100 ;*** routine: diskette initialisieren ***
110 ;
120 .ba 49152    ;* startadresse *
130 ;
140          lda #$0f       ;logische file-nr. 15
150          ldx #$08       ;geraeteadresse
160          ldy #$0f       ;sekundaeradr. 15
170          jsr $ffba      ;parameter setzen
180 ;
190          lda #$01       ;befehlslaenge!
200          ldx #<(befehl) ;zeiger auf
210          ldy #>(befehl) ;befehlstext
220          jsr $ffbd      ;parameter setzen
230 ;
240          jsr $ffc0      ;open-routine aufrufen
250 ;
260          lda #$0f       ;logische file-nr. 15
270          jsr $ffc3      ;close-routine aufrufen
280 ;
290 befehl   .tx "i"
```

Disketten validieren

Bei aller Sorgfalt, mit der die Floppy natürlich arbeitet, kann es mitunter vorkommen, daß ein Datenblock in der BAM als belegt gekennzeichnet ist, obwohl er überhaupt keiner Datei mehr zugeordnet ist.

Die häufigste Ursache dafür liegt in gelöschten Dateien. Beim Löschen einer Datei wird diese nämlich nur im Directory der Diskette als "gelöscht" gekennzeichnet: Ihre Datenblocks bleiben zunächst "belegt". Dadurch wird sozusagen eine "Hintertür" offengehalten, um die Datei wieder zurückzuholen (indem man den Löscheintrag im Directory zum Beispiel mit einem sogenannten Diskettenmonitor rückgängig macht). Das funktioniert allerdings nur so lange, wie man keine neuen Daten auf der betreffenden Diskette speichert.

Um nun aber die BAM gegebenenfalls auf den neuesten Stand zu bringen, besteht die Möglichkeit, die Diskette "aufräumen" zu lassen. Diesen Vorgang bezeichnet man auch als "Validieren". Der Befehl dafür schreibt sich analog:

```
VALIDATE
```

oder abgekürzt:

```
V
```

Beim Validieren überprüft die Floppy jeden der 664 Datenblöcke der Diskette. Ist ein Block keiner im Directory eingetragenen Datei zugeordnet, so wird er als frei gekennzeichnet. In BASIC sieht das so aus:

```
100 rem *** routine: diskette validieren ***
110 open 15,8,15
120 print#15,"v"
130 close 15
```

Und das ganze in Assembler:

```
100 ;*** routine: diskette validieren ***
110 ;
120 .ba 49152    ;* startadresse *
130 ;
140           lda #$0f       ;logische file-nr. 15
150           ldx #$08       ;geraeteadresse
160           ldy #$0f       ;sekundaeradr. 15
170           jsr $ffba      ;parameter setzen
180 ;
190           lda #$01       ;befehlslaenge!
200           ldx #<(befehl) ;zeiger auf
210           ldy #>(befehl) ;befehlstext
220           jsr $ffbd      ;parameter setzen
230 ;
240           jsr $ffc0      ;open-routine aufrufen
250 ;
260           lda #$0f       ;logische file-nr. 15
270           jsr $ffc3      ;close-routine aufrufen
280 ;
290 befehl    .tx "v"
```

Dateien löschen

Eine Datei, die Sie nicht mehr benötigen, können Sie auf der Diskette jederzeit löschen. Dazu gibt es den Befehl SCRATCH:

```
"SCRATCH:DATEINAME"
```

oder abgekürzt:

```
"S:DATEINAME"
```

In BASIC sieht das so aus:

```
100 rem *** routine: datei loeschen ***
110 open 15,8,15
120 print#15,"s:daten(alt)"
130 close 15
```

Und das ganze in Assembler:

```
100 ;*** routine: datei loeschen ***
110 ;
120 .ba 49152   ;* startadresse *
130 ;
140          lda #$0f       ;logische file-nr. 15
150          ldx #$08       ;geraeteadresse
160          ldy #$0f       ;sekundaeradr. 15
170          jsr $ffba      ;parameter setzen
180 ;
190          lda #$0c       ;befehlslaenge!
200          ldx #<(befehl) ;zeiger auf
210          ldy #>(befehl) ;befehlstext
220          jsr $ffbd      ;parameter setzen
230 ;
240          jsr $ffc0      ;open-routine aufrufen
250 ;
260          lda #$0f       ;logische file-nr. 15
270          jsr $ffc3      ;close-routine aufrufen
280 ;
290 befehl   .tx "s:daten(alt)"
```

Mit einem SCRATCH-Befehl lassen sich auch mehrere Dateien gleichzeit löschen. Dazu gibt man die Dateinamen hintereinander an, beispielsweise löscht

```
OPEN 15,8,15,"S:DATEI1,DATEI2,DATEI3":CLOSE 15
```

drei Dateien auf einmal. Die Länge des Befehls-Strings ist allerdings auf 40 Zeichen beschränkt. Bei ähnlichen Dateinamen kann man sich auch der beiden sogenannten Jokerzeichen * und ? bedienen.

Das Fragezeichen ? signalisiert der Floppy, daß das Zeichen an der entsprechenden Stelle beliebig sein darf.

```
OPEN 15,8,15,"S:DATEI?":CLOSE 15
```

löscht daher ebenfalls die drei Dateien, allerdings auch evtl. vorhandene Dateien mit den Namen DATEI4, DATEI5 usw.! Der Stern * kennzeichnet den Rest des Dateinamens als beliebig.

```
OPEN 15,8,15,"S:DAT*":CLOSE 15
```

löscht also sämtliche Dateien, die mit "DAT" beginnen.

Vorsicht: Eine Datei, die mit SCRATCH gelöscht wurde, kann - zumindest mit normalen Mitteln - nicht mehr zurückgeholt werden! Insbesondere bei der Arbeit mit den Jokerzeichen sollten Sie daher sehr sorgfältig darauf achten, daß Sie nicht aus Versehen eine falsche Datei löschen.

Wenn Sie nach dem Löschen den Fehlerkanal auslesen, erhalten Sie die Anzahl der gelöschten Dateien.

```
03 FILES SCRATCHED 00 00
```

beispielsweise besagt, daß drei Dateien gelöscht wurden.

Dateien umbenennen

Wesentlich "ungefährlicher" als das Löschen ist das Umbenennen einer Datei. Dazu stellt die Floppy einen Befehl zur Verfügung:

```
RENAME:NEUER NAME=ALTER NAME
```

oder abgekürzt:

```
R:NEUER NAME=ALTER NAME
```

Möchten Sie beispielsweise die Datei DATEN in DATEN(ALT) umbenennen, so sieht das in BASIC wie folgt aus:

```
100 rem *** routine: datei umbenennen ***
110 open 15,8,15
120 print#15,"r:daten(alt)=daten"
130 close 15
```

Und das ganze in Assembler:

```
100 ;*** routine: datei umbenennen ***
110 ;
120 .ba 49152    ;* startadresse *
130 ;
140          lda #$0f        ;logische file-nr. 15
150          ldx #$08        ;geraeteadresse
160          ldy #$0f        ;sekundaeradr. 15
170          jsr $ffba       ;parameter setzen
180 ;
190          lda #$12        ;befehlslaenge!
200          ldx #<(befehl) ;zeiger auf
210          ldy #>(befehl) ;befehlstext
220          jsr $ffbd       ;parameter setzen
230 ;
240          jsr $ffc0       ;open-routine aufrufen
250 ;
260          lda #$0f        ;logische file-nr. 15
270          jsr $ffc3       ;close-routine aufrufen
280 ;
290 befehl   .tx "r:daten(alt)=daten"
```

Dateien kopieren

Ein Befehl, der auf den ersten Blick keinen allzu großen Sinn macht, ist COPY. COPY kopiert eine Datei innerhalb einer Diskette. Nun möchte man ja aber in der Regel allenfalls eine Kopie einer Datei auf einer anderen Diskette haben und keine zwei Dateien mit demselben Inhalt auf einer Diskette.

COPY hat aber durchaus seine Existenzberechtigung. Mit ihm lassen sich nämlich auch mehrere sogenannte "sequentielle Dateien" (mehr dazu im nächsten Abschnitt) zu einer neuen Datei zusammenfassen:

```
COPY:NEUEDATEI=DATEI1,DATEI2,DATEI3,...
```

oder abgekürzt:

```
C:NEUEDATEI=DATEI1,DATEI2,DATEI3,...
```

Die hinter dem Gleichheitszeichen stehenden Dateien werden zu der Datei NEUEDATEI zusammengefaßt. Die Einzeldateien sind natürlich auch anschließend noch verfügbar!

In BASIC könnte das so aussehen:

```
100 rem *** routine: dateien zusammenfügen ***
110 open 15,8,15
120 print#15,"c:jahr=quart1,quart2,quart3,quart4"
130 close 15
```

Und das ganze in Assembler:

```
100 ;*** routine: dateien zusammenfügen ***
110 ;
120 .ba 49152     ;* startadresse *
130 ;
140           lda #$0f        ;logische file-nr. 15
150           ldx #$08        ;geraeteadresse
160           ldy #$0f        ;sekundaeradr. 15
170           jsr $ffba       ;parameter setzen
180 ;
190           lda #$22        ;befehlslaenge!
200           ldx #<(befehl)  ;zeiger auf
210           ldy #>(befehl)  ;befehlstext
220           jsr $ffbd       ;parameter setzen
230 ;
240           jsr $ffc0       ;open-routine aufrufen
250 ;
260           lda #$0f        ;logische file-nr. 15
270           jsr $ffc3       ;close-routine aufrufen
280 ;
290 befehl    .tx "c:jahr=quart1,quart2,quart3,quart4"
```

Die vier Quartalsdateien QUART1 bis QUART4 (die vielleicht irgendwelche Quartalsdaten enthalten) werden zu der Jahresdatei JAHR zusammengefügt.

5.2 Sequentielle Dateien

Die einfachste Form der Datenspeicherung findet in sogenannten sequentiellen Dateien statt. Jede Datei ist dabei in einzelne Datensätze unterteilt. Jeder Datensatz wiederum besteht aus einer bestimmten Anzahl von Datenfeldern:

D a t e i								
Datensatz 1			Datensatz 2			Datensatz 3		
F.1	F.2	F.3	F.1	F.2	F.3	F.1	F.2	F.3

Wie aus der Abbildung ersichtlich ist, werden die einzelnen Datensätze in der Datei hintereinander - als eine Sequenz - abgelegt. Daher kommt auch der Name sequentielle Datei. Eine sequentielle Datei kann nur komplett geschrieben oder gelesen werden. Ein direkter Zugriff auf einen bestimmten Datensatz ist nicht möglich, dafür dürfen die einzelnen Datensätze aber eine unterschiedliche Länge haben.

Nehmen wir als Beispiel einmal eine Adreßverwaltung, die vielleicht aus 100 Datensätzen besteht. In jedem Datensatz ist eine Adresse abgelegt, die aus fünf Teilen besteht:

```
Nachname
Vorname
Straße
Ort
Telefon
```

Nachname, Vorname usw. sind die Datenfelder eines Datensatzes. Um diese Adressen nun zu verwalten, wird die gesamte Datei am Programmanfang in den Rechnerspeicher (in ein Variablenfeld) geholt.

Dadurch ergibt sich schon eine wichtige Einschränkung, was die Datenmenge betrifft: Sie ist durch den verfügbaren Rechnerspeicher beschränkt. Dafür stehen die Daten im Rechner aber extrem schnell zur Verfügung. So kann man sich beispielsweise

ganz einfach blitzschnell auch eine ganz bestimmte Adresse am Bildschirm ausgeben lassen.

Am Ende des Programms wird die komplette Datei wieder auf die Diskette zurückgeschrieben. Das ist natürlich nur dann erforderlich, wenn Sie einzelne Adressen geändert oder neue hinzugefügt haben. Damit ist das Grundprinzip der sequentiellen Datei auch schon erklärt. Die sequentielle Datenspeicherung zählt zwar nicht zu den schnellsten, dafür aber mit Sicherheit zu den einfachsten Arten der Datenverwaltung.

Gehen wir nun die einzelnen Programmierschritte der Reihe nach durch.

Eine sequentielle Datei öffnen

Als erstes muß die Datei geöffnet werden. Durch das Öffnen der Datei signalisieren Sie dem Rechner und der Floppy, daß Sie auf die Datei zugreifen wollen. Der OPEN-Befehl ist wie folgt aufgebaut:

```
OPEN LF,8,SA,"Dateiname,S,Modus"
```

LF ist die logische File-Nummer, über die Sie sich bei späteren Lese- oder Schreibzugriffen auf die Datei beziehen können. Für LF können Sie Werte zwischen 1 und 127 wählen. Bei der 8 handelt es sich um die Geräteadresse der Floppy.

Die Sekundäradresse SA hat für die Floppy eine ähnliche Bedeutung wie die logische File-Nummer für den Rechner. Anhand der Sekundäradresse erkennt die Floppy, auf welche Datei Sie bei einem Schreib-/Lesezugriff jeweils zugreifen wollen.

Falls Sie daher mehrere sequentielle Dateien gleichzeitig öffnen wollen (bis zu drei sind erlaubt), ist es sehr wichtig, daß diese verschiedene Sekundäradressen haben!

Für SA sind nur Werte zwischen 2 und 14 zugelassen. Die Werte 0 und 1 sind für das Speichern und Laden von Programmen reserviert, die 15, wie Sie ja bereits wissen, für den Befehlskanal.

Am einfachsten ist es, wenn man für die logische File-Nummer und die Sekundäradresse jeweils denselben Wert nimmt. Möchte man die Maximalzahl von drei Dateien gleichzeitig öffnen, dann sähe das so aus:

```
OPEN 2,8,2,...
OPEN 3,8,3,...
OPEN 4,8,4,...
```

Wenn Sie sich an diese einfachen Werte halten, verlieren Sie nicht so schnell den Überblick und vermeiden meist nur sehr schwer zu entdeckende Fehler bei späteren Schreib- und Lesezugriffen. Die drei Parameter in den Anführungszeichen sind schnell erklärt. Der Dateiname dürfte klar sein. Er darf bis zu 16 Zeichen lang sein. Das "S" steht für den Dateityp, also sequentiell.

"Modus" bezeichnet die Art des Zugriffs (Lesen oder Schreiben). Sie müssen also bereits beim Öffnen der Datei festlegen, ob Sie aus ihr Daten lesen (Modus "R" für "Read") oder in sie Daten schreiben (Modus "W" für "Write") wollen.

```
OPEN 2,8,2,"DATEN,S,W"
```

öffnet die Datei DATEN also zum Schreiben. Falls auf der eingelegten Diskette bereits eine Datei mit diesem Namen existiert (egal von welchem Dateityp), erhalten Sie die Fehlermeldung FILE EXISTS. In diesem Fall müssen Sie entweder einen anderen Namen wählen oder die vorhandene Datei zuvor mit

```
OPEN 15,8,15,"S:DATEN":CLOSE 15
```

löschen.

```
OPEN 2,8,2,"DATEN,S,R"
```

öffnet die Datei DATEN zum Lesen. Falls auf der Diskette keine sequentielle Datei dieses Namens existiert, bekommen Sie die Fehlermeldung FILE NOT FOUND. In Assembler sieht das ganze ähnlich aus:

```
100 ;*** routine: sequentielle datei öffnen ***
110 ;
120 .ba 49152   ;* startadresse *
130 ;
140           lda #$02        ;logische file-nr. 2
150           ldx #$08        ;geraeteadresse
160           ldy #$02        ;sekundaeradr. 2
170           jsr $ffba       ;parameter setzen
180 ;
190           lda #$0d        ;textlaenge!
200           ldx #<(text)    ;zeiger auf
210           ldy #>(text)    ;text
220           jsr $ffbd       ;parameter setzen
230 ;
240           jsr $ffc0       ;open-routine aufrufen
250 ;
260           ldx #$02        ;logische file-nr. 2
270           jsr $ffc6       ;datei auf eingabe schalten
280 ;
290 text      .tx "dateiname,s,r"
```

Um in die Datei zu schreiben, müssen Sie die letzten drei Programmzeilen austauschen:

```
270           jsr $ffc9       ;datei auf ausgabe schalten
280 ;
290 text      .tx "dateiname,s,w"
```

Eine sequentielle Datei schließen

Nachdem die Daten gelesen oder geschrieben wurden, muß die Datei wieder geschlossen werden. In BASIC verwenden Sie dazu den CLOSE-Befehl:

```
CLOSE LF
```

LF ist die logische File-Nummer, die Sie beim Öffnen der Datei angegeben haben. In Assembler sind drei Befehle erforderlich:

```
500           jsr $ffcc       ;auf standardeingabe
                              ;zurückschalten
510           lda #$02        ;logische file-nr. 2
520           jsr $ffc3       ;close-routine aufrufen
```

Das Schließen der Datei signalisiert dem Rechner und der Floppy, daß die Arbeit mit dieser Datei beendet ist. Insbeson-

dere in bezug auf die Floppy ist es daher sehr wichtig, daß Sie auch das Schließen nicht vergessen.

Wenn Sie beispielsweise eine sequentielle Datei zum Schreiben geöffnet haben und vergessen, sie zu schließen, dann war die ganze Mühe umsonst. Die Datei läßt sich anschließend nämlich nicht mehr lesen! Bei jedem Öffnungsversuch zum Lesen erhalten Sie nur die Fehlermeldung WRITE FILE OPEN.

Besonders tückisch wird das ganze beim Austesten von Programmen, die mit Floppy-Dateien arbeiten. Wenn Ihr Programm beispielsweise infolge eines Syntaxfehlers irgendwo abbricht, sollten Sie zuallererst nachsehen, ob im Moment noch eine Datei geöffnet ist, und falls ja, diese sofort mit

```
CLOSE LF
```

schließen. Sobald Sie nämlich den Syntaxfehler in Ihrem Programm behoben haben, ist es zu spät, da der Rechner beim Eingeben einer neuen Programmzeile automatisch alle internen Vermerke über geöffnete Dateien löscht. Die Datei ist dann zwar für den Rechner "geschlossen", nicht aber für die Floppy! Eine nicht geschlossene Datei ist übrigens bei der Floppy leicht "optisch" erkennbar: Sobald eine Datei (egal welchen Typs) geöffnet wird, beginnt die rote LED der Floppy zu leuchten und erlischt erst wieder, nachdem die letzte Datei geschlossen ist.

Daten schreiben und lesen

Kommen wir zum wichtigsten, dem Schreiben und Lesen von Daten. Dazu benötigen wir die drei BASIC-Befehle PRINT#, INPUT# und GET#. Nehmen wir einmal an, Sie möchten in die Datei ADRESSEN1 den Datensatz

```
Klaus
Mueller
Beispielstr. 29
8000 Muenchen 60
0811/12 34 56
```

schreiben, dann gehen Sie wie folgt vor:

```
100 open 2,8,2,"adressen1,s,w"
110 print#2,"klaus"
120 print#2,"mueller"
130 print#2,"beispielstr. 29"
140 print#2,"8000 muenchen 60"
150 print#2,"0811/12 34 56"
160 close 2
```

Wieder einlesen läßt sich der Datensatz ebenso leicht:

```
200 open 2,8,2,"adressen1,s,r"
210 input#2,vn$
220 input#2,nn$
230 input#2,sr$
240 input#2,ot$
250 input#2,tf$
260 close 2
270 :
280 print "vorname:  ";vn$
290 print "nachname: ";nn$
300 print "strasse:  ";sr$
310 print "ort:"     ";ot$
320 print "telefon:  ";tf$
```

Auf dem Bildschirm erhalten Sie folgende Ausgabe:

```
VORNAME:  KLAUS
NACHNAME: MUELLER
STRASSE:  BEISPIELSTR. 29
ORT:      8000 MUENCHEN 6
TELEFON:  0811/12 34 56
```

Damit der INPUT#-Befehl die einzelnen Datenfelder auseinanderhalten kann, müssen diese entweder durch ein "Return" (ASCII-Code 13) oder ein Komma (,) voneinander getrennt sein.

Das "Return" wird vom PRINT#-Befehl am Ende jeder Zeile automatisch gesendet. Wohlgemerkt aber nur am Ende! Wollen Sie mehrere Datenfelder in einer PRINT#-Zeile zusammenfassen, dann müssen Sie selbst für die Feldertrennung sorgen, indem Sie ein Komma zwischen die Felder setzen:

```
100 open 2,8,2,"adressen1,s,w"
110 print#2,"klaus";",";"mueller";",";"beispielstr. 29"
140 print#2,"8000 muenchen 60";",";"0811/12 34 56"
160 close 2
```

Die Felder 1,2 und 3 sowie 4 und 5 sind nun durch ein Komma getrennt, die Felder 3 und 4 durch ein "Return". Natürlich können Sie auch innerhalb einer Zeile "Return" einfügen:

```
100 open 2,8,2,"adressen1,s,w"
110 print#2,"klaus";chr$(13);"mueller";chr$(13);"beispielstr. 29"
140 print#2,"8000 muenchen 60";chr$(13);"0811/12 34 56"
160 close 2
```

Jeden Datensatz durch einen einzelnen PRINT#-Befehl zu schreiben, ist sehr umständlich. Wesentlich einfacher wird das ganze mit einer Einleseschleife und fünf eindimensionalen Variablenfeldern:

```
100 input "anzahl der datensaetze:";az
110 rem felder dimensionieren
120 dim vn$(az),nn$(az),sr$(az),ot$(az),tf$(az)
130 :
140 rem datensaetze erfragen
150 for z=1 to az
160 input "vorname:  ";vn$(z)
170 input "nachname: ";nn$(z)
180 input "strasse:  ";sr$(z)
190 input "ort:"     ";ot$(z)
200 input "telefon:  ";tf$(z)
210 next z
220 :
230 rem datensaetze in datei schreiben
240 open 2,8,2,"adressen2,s,w"
250 for z=1 to az
260 print#2,vn$(z)
270 print#2,nn$(z)
280 print#2,sr$(z)
290 print#2,ot$(z)
300 print#2,tf$(z)
310 next z
320 close 2
```

Das Programm erlaubt es Ihnen, eine beliebige Anzahl von Datensätzen einzugeben, die dann in der Datei ADRESSEN2 abgelegt werden. Um es noch etwas flexibler zu gestalten, könnte

man die Anzahl der Datensätze in der Datei mit abspeichern lassen. Dazu fügen Sie einfach eine Programmzeile 245 ein:

```
245 print#2,az
```

Mit dem folgenden Programm, das die Daten wieder einliest, haben Sie schon fast eine kleine Adreßverwaltung:

```
500 rem datensaetze einlesen
510 open 2,8,2,"adressen2,s,r"
520 input az:rem anzahl der datensaetze
530 dim vn$(az),nn$(az),sr$(az),ot$(az),tf$(az):rem felder
540 for z=1 to az
550 input#2,vn$(z)
560 input#2,nn$(z)
570 input#2,sr$(z)
580 input#2,ot$(z)
590 input#2,tf$(z)
600 next z
610 close 2
620 :
630 rem datensaetze ausgeben
640 for z=1 to az
650 print "vorname:  ";vn$(z)
660 print "nachname: ";nn$(z)
670 print "strasse:  ";sr$(z)
680 print "ort:"     ";ot$(z)
690 print "telefon:  ";tf$(z)
700 print
710 print "bitte eine taste druecken!"
720 get tt$:if tt$="" then 720:rem auf tastendruck warten
730 next z
```

Insbesondere dann, wenn man eine fremde Datei einlesen möchte, stellt sich oft das Problem, daß man den internen Aufbau der Datei nicht genau kennt, also nicht weiß, aus wie vielen Datensätzen die Datei besteht und wie groß die einzelnen Datensätze jeweils sind.

In diesem Fall nützt einem der INPUT#-Befehl nur wenig. Nicht zuletzt, da die Eingabelänge bei INPUT# auf 88 Zeichen beschränkt ist. Dafür gibt es aber noch einen anderen Eingabebefehl, der jeweils nur ein einzelnes Zeichen einliest: GET#.

Was wir außerdem noch benötigen, ist die Möglichkeit, festzustellen, wann das Ende einer Datei erreicht ist. Da es ja nicht

gesagt ist, daß jede Datei mit einem "Return", das wir leicht abfragen können, endet, bedienen wir uns der sogenannten "reservierten Variablen" ST.

Enthält ST einen Wert ungleich 64, so können wir aus der aktuellen Datei unbesorgt weiter Daten einlesen. Das Dateiende ist dann noch nicht erreicht.

Nebenbei bemerkt: Da die Variable ST, wie gesagt, vom Betriebssystem "reserviert" ist, darf man sie nicht für eigene Zwecke verwenden. Wenn Sie das versuchen, beispielsweise durch eine Wertzuweisung "ST=...", erhalten Sie eine entsprechende Fehlermeldung. Das folgende kleine Programm ist in der Lage, eine beliebige sequentielle Datei zu lesen:

```
100 open 2,8,2,"datei,s,r"
110 :
120 if st=64 then 210:rem dateiende erreicht?
130 get#2,eg$:rem ein zeichen holen
140 if eg$=chr$(13) then 180:rem 'return'?
150 dt$=dt$+eg$
160 goto 120:rem naechstes zeichen
170 :
180 print dt$
190 dt$="":goto 120:rem naechster datensatz
200 :
210 close 2
```

Die einzelnen Datensätze werden nacheinander in die Variable DT$ eingelesen und auf dem Bildschirm ausgegeben.

Die einzige Bedingung dabei: Ein Datensatz darf nicht länger als 255 Zeichen sein, da ja DT$ keine längeren Zeichenketten aufnehmen kann. Kennt man die Länge der Datensätze bzw. der einzelnen Felder, dann genügt eine einfache FOR-NEXT-Schleife:

```
100 open 2,8,2,"datei,s,r"
110 dt$=""
120 for z=1 to 150
130 get#2,eg$
140 dt$=dt$+eg$
150 next z
.....
```

Das Programmstück liest ein 150 Byte langes Datenfeld ein und legt es in der Variablen DT$ ab. Bevor man jetzt das nächste Feld einliest, darf man aber nicht vergessen, das trennende "Return" zu überlesen! Das Programmstück müßte also wie folgt erweitert werden:

```
160 get eg$:rem 'return' ueberlesen
165 rem naechstes datenfeld
170 nt$=""
180 for z=1 to ...
190 get#2,eg$
200 nt$=nt$+eg$
210 next z
.....
```

In Assembler möchte ich mich der Einfachheit halber auf das Schreiben und Lesen eines Datensatzes beschränken. Dabei gehe ich davon aus, daß sich der zu schreibende Datensatz in einem Pufferspeicher befindet (Programmzeile 580) und mit einem "Return" abgeschlossen ist:

```
100 ;*** routine: datensatz schreiben ***
110 ;
120 .ba 49152    ;* startadresse *
125 ;
130 ;
135 ;* datei oeffnen *
140           lda #$02       ;logische file-nr. 2
150           ldx #$08       ;geraeteadresse
160           ldy #$02       ;sekundaeradr. 2
170           jsr $ffba      ;parameter setzen
180 ;
190           lda #$0d       ;textlaenge!
200           ldx #<(text)   ;zeiger auf
210           ldy #>(text)   ;text
220           jsr $ffbd      ;parameter setzen
230 ;
240           jsr $ffc0      ;open-routine aufrufen
250 ;
260           ldx #$02       ;logische file-nr. 2
270           jsr $ffc9      ;datei auf ausgabe schalten
280 ;
285 ;
290 ;* datensatz schreiben *
300           ldy #0
310 schleife  lda puffer,y   ;zeichen aus puffer holen
320           jsr $ffd2      ;und ausgeben
330           cmp #$0d       ;'return'?
340           beq fertig     ;ja, ausgabe beenden
```

```
350            iny            ;zeiger+1
360            jmp schleife   ;naechstes zeichen
370 ;
380 ;
500 ;* datei schliessen *
510 fertig     jsr $ffcc      ;auf standardeingabe
                              ;zurückschalten
520            lda #$02       ;logische file-nr. 2
530            jsr $ffc3      ;close-routine aufrufen
540 ;
550 ;
560 ;* datenbereich *
570 text       .tx "dateiname,s,w"
580 puffer     .tx "...datensatz...                  "
```

Der zu lesende Datensatz wird in dem in Programmzeile 580 eingerichteten Puffer abgelegt. Dieser Puffer muß natürlich ausreichend groß dimensioniert werden:

```
100 ;*** routine: datensatz lesen ***
110 ;
120 .ba 49152    ;* startadresse *
125 ;
130 ;
135 ;* datei oeffnen *
140            lda #$02       ;logische file-nr. 2
150            ldx #$08       ;geraeteadresse
160            ldy #$02       ;sekundaeradr. 2
170            jsr $ffba      ;parameter setzen
180 ;
190            lda #$0d       ;textlaenge!
200            ldx #<(text)   ;zeiger auf
210            ldy #>(text)   ;text
220            jsr $ffbd      ;parameter setzen
230 ;
240            jsr $ffc0      ;open-routine aufrufen
250 ;
260            ldx #$02       ;logische file-nr. 2
270            jsr $ffc6      ;datei auf eingabe schalten
280 ;
285 ;
290 ;* datensatz lesen *
300            ldy #0
310 schleife   jsr $ffcf      ;ein zeichen lesen
320            cmp #$0d       ;'return'?
330            beq fertig     ;ja, eingabe beenden
340            sta puffer,y   ;zeichen in puffer ablegen
350            iny            ;zeiger+1
360            jmp schleife   ;naechstes zeichen
370 ;
380 ;
```

```
500 ;* datei schliessen *
510 fertig    jsr $ffcc      ;auf standardeingabe
                             ;zurückschalten
520           lda #$02       ;logische file-nr. 2
530           jsr $ffc3      ;close-routine aufrufen
540 ;
550 ;
560 ;* datenbereich *
570 text      .tx "dateiname,s,r"
580 puffer    .tx "                          "
```

Sequentielle Dateien erweitern

Ein nicht zu unterschätzender Nachteil der sequentiellen Datenspeicherung ist die mangelnde Flexibilität, wenn es um Änderungen an den Daten geht. Im Normalfall muß die gesamte Datei eingelesen, die Änderungen durchgeführt und anschließend die Datei wieder zurückgeschrieben werden.

Falls Sie die Datei aber nur um zusätzliche Datensätze erweitern wollen und diese Datensätze auch noch "hinten" an die Datei angehängt werden sollen, gibt es einen wesentlich einfacheren Weg. Dazu müssen Sie die Datei in einem speziellen Modus, dem Append-Modus, öffnen:

```
OPEN 2,8,2,"DATEN,S,A"
```

Wenn Sie anschließend mit

```
PRINT#2,"NEUE DATEN"
```

in die Datei schreiben, werden die zusätzlichen Daten automatisch an die bereits in der Datei vorhandenen Daten angehängt. Zum Schluß dürfen Sie natürlich nicht vergessen, die Datei mit

```
CLOSE 2
```

zu schließen. Nehmen wir ein einfaches Beispiel: Zuerst schreiben wir in eine neue Datei mit dem Namen BEISPIEL fünf Datensätze:

```
100 open 2,8,2,"beispiel,s,w"
110 print#2,"datensatz 1"
120 print#2,"datensatz 2"
```

```
130 print#2,"datensatz 3"
140 print#2,"datensatz 4"
150 print#2,"datensatz 5"
160 close 2
```

Anschließend hängen wir drei Datensätze an:

```
200 open 2,8,2,"beispiel,s,a"
210 print#2,"datensatz 6"
220 print#2,"datensatz 7"
230 print#2,"datensatz 8"
240 close 2
```

Die Datei BEISPIEL enthält nun acht Datensätze, die Sie mit dem folgenden Programmstück einlesen können:

```
300 open 2,8,2,"beispiel,s,r"
310 for z=1 to 8
320 input#2,dz$
330 print dz$
340 next z
350 close 2
```

In Assembler möchte ich mich auf einen Datensatz beschränken:

```
100 ;*** routine: datensatz anhaengen ***
110 ;
120 .ba 49152    ;* startadresse *
125 ;
130 ;
135 ;* datei oeffnen *
140           lda #$02       ;logische file-nr. 2
150           ldx #$08       ;geraeteadresse
160           ldy #$02       ;sekundaeradr. 2
170           jsr $ffba      ;parameter setzen
180 ;
190           lda #$0d       ;textlaenge!
200           ldx #<(text)   ;zeiger auf
210           ldy #>(text)   ;text
220           jsr $ffbd      ;parameter setzen
230 ;
240           jsr $ffc0      ;open-routine aufrufen
250 ;
260           ldx #$02       ;logische file-nr. 2
270           jsr $ffc9      ;datei auf ausgabe schalten
280 ;
285 ;
290 ;* datensatz schreiben (anhaengen) *
300           ldy #0
310 schleife  lda puffer,y   ;zeichen aus puffer holen
```

```
320            jsr $ffd2      ;und ausgeben
330            cmp #$0d       ;'return'?
340            beq fertig     ;ja, ausgabe beenden
350            iny            ;zeiger+1
360            jmp schleife   ;naechstes zeichen
370 ;
380 ;
500 ;* datei schliessen *
500 fertig     jsr $ffcc      ;auf standardeingabe
                              ;zurückschalten
510            lda #$02       ;logische file-nr. 2
520            jsr $ffc3      ;close-routine aufrufen
520 ;
530 ;
530 ;* datenbereich *
290 text       .tx "dateiname,s,a"
540 puffer     .tx "...datensatz...               "
```

Nicht zuletzt wegen der doch recht "starren" Struktur der sequentiellen Dateien hat man eine weitere Dateiform eingeführt, die sogenannten "relativen Dateien", mit denen wir uns im nächsten Abschnitt befassen werden.

5.3 Relative Dateien

Eine sogenannte relative Datei ist in Grunde genommen nicht viel anders aufgebaut als eine sequentielle Datei. Der entscheidende Unterschied liegt in der Art des Zugriffs auf die einzelnen Datensätze. Bei einer relativen Datei kann man jeden Datensatz direkt ansprechen, also schreiben oder lesen. D.h., es muß nicht extra die gesamte Datei in den Rechnerspeicher geladen werden, sondern eben nur der jeweils benötigte Datensatz.

Der Vorteil liegt klar auf der Hand: Die Datenkapazität einer relativen Datei ist nur durch die Größe einer Diskette begrenzt und nicht wie bei der sequentiellen Datei durch die Größe des Rechnerspeichers.

Außerdem gestaltet sich die Datenverwaltung mit einer relativen Datei natürlich wesentlich flexibler. Möchte man beispielsweise einen bestimmten Datensatz (von vielleicht 1000) ändern, so muß man nur diesen einen Satz lesen und - nach den durchgeführten Änderungen - wieder zurückschreiben.

Wie so oft hat das ganze aber auch einen Haken: alle Datensätze einer relativen Datei (auch Records genannt) müssen dieselbe Länge aufweisen. Diese Einschränkung ergibt sich aus der Art und Weise, wie die Floppy die Position eines Datensatzes innerhalb der Gesamtdatei ermittelt. Nehmen wir einmal an, Sie haben eine relative Datei mit 500 Datensätzen, wobei jeder Datensatz 110 Byte lang ist. Nun möchten Sie den 137. Datensatz lesen. Die Floppy führt jetzt eine einfache Rechnung durch:

```
Datensatzlänge*(Datensatznummer-1)=Position
```

Da Sie den 137. Datensatz benötigen, müssen die ersten 136 Datensätze "überlesen" werden. Wegen der konstanten Datensatzlänge von 110 Byte belegen diese 136 Sätze genau 110*136=14960 Byte in der relativen Datei. Der 137. Datensatz beginnt also ab der Position 14961! Auf diese Stelle innerhalb der Datei setzt die Floppy einen Zeiger, so daß Sie den Datensatz anschließend lesen oder schreiben können.

Eine relative Datei öffnen und schließen

Zuerst muß die Datei aber geöffnet werden. Das geschieht, wie gewohnt, mit dem OPEN-Befehl. Im Gegensatz zu den sequentiellen Dateien ist es dabei aber egal, ob Sie in die Datei schreiben oder aus ihr lesen möchten. Der OPEN-Befehl hat das folgende Format:

```
OPEN LF,8,SA,"DATEINAME,L,"+CHR$(RL)
```

bzw. in Assembler:

```
140          lda #$02        ;logische file-nr. 2
150          ldx #$08        ;geraeteadresse
160          ldy #$02        ;sekundaeradr. 2
170          jsr $ffba       ;parameter setzen
180 ;
190          lda #$0d        ;textlaenge!
200          ldx #<(text)    ;zeiger auf
210          ldy #>(text)    ;text
220          jsr $ffbd       ;parameter setzen
230 ;
240          jsr $ffc0       ;open-routine aufrufen
```

```
280 ;
290 text      .tx "dateiname,l,"
300           .by rl
```

LF ist wiederum die logische File-Nummer der Datei; SA ihre Sekundäradresse. Das "L" hinter dem Dateinamen signalisiert der Floppy, daß Sie eine relative Datei öffnen wollen. Übrigens kann immer nur eine relative Datei geöffnet sein. Möchte man mit mehreren relativen Dateien gleichzeitig arbeiten, muß man die Dateien vor jedem Zugriff öffnen und anschließend sofort wieder schließen. Eine zusätzlich geöffnete sequentielle Datei ist aber erlaubt.

RL steht für die Record-Länge. Die Record-Länge kann zwischen 1 und 254 Byte gewählt werden und muß bei jedem Öffnen der Datei mit angegeben werden. Eine nachträgliche Änderung der Record-Länge ist nicht möglich. Wenn Sie bei einem späteren Zugriff auf die Datei eine andere Record-Länge als beim Einrichten der Datei angeben, reagiert die Floppy mit einer Fehlermeldung.

Bevor man eine relative Datei einrichtet, muß man sich daher sehr sorgfältig überlegen, wie groß ein Datensatz maximal werden kann. Dabei spielt es auch eine Rolle, wie Sie den Datensatz bzw. die einzelnen Datenfelder später einlesen wollen.

Wenn sie mit INPUT# arbeiten, müssen die Records bzw. die einzelnen Datenfelder mit einem "Return" abgeschlossen sein. Dieses "Return" belegt aber ein Byte des Records. Falls die Datensätze also beispielsweise 50 Bytes groß sind, müssen Sie als Record-Länge 51 Bytes angeben.

Eine Alternative ist der GET#-Befehl, der ja immer nur ein Zeichen holt. Der große Nachteil von GET#: Es ist insbesondere bei langen Records viel zu langsam. Ich werde Ihnen deshalb etwas weiter unten eine Maschinensprache-Routine vorstellen, mit der sich Records bis zur Maximallänge von 254 Bytes in einem Stück in eine Variable einlesen lassen.

Das Öffnen der Datei haben wir nun hinter uns. Doch wie geht es weiter? Im Augenblick enthält die Datei ja noch kein einziges Byte, geschweige denn einen kompletten Datensatz. An dieser Stelle kommt die eingangs schon erwähnte Record-Positionierung ins Spiel. Um die Datei voll einzurichten, setzen wir den Record-Zeiger einfach auf den letzten Record, den wir voraussichtlich benötigen werden, und beschreiben diesen Record mit dem ASCII-Code 255. Wenn Sie also eine relative Datei mit 200 Datensätzen einrichten wollen, positionieren Sie auf den 200. Record. Dadurch werden automatisch auch alle vor diesem Datensatz liegenden Records eingerichtet oder "freigegeben", wie man auch sagt.

Diese Festlegung ist im Gegensatz zur Record-Länge aber nicht endgültig. Sie können die Datei jederzeit um zusätzliche Datensätze erweitern, indem Sie einfach auf einen höheren Record positionieren und diesen mit CHR$(255) beschreiben.

Wenn Sie dagegen versuchen, einen Record zu lesen, der über dem momentanen Dateiende liegt, erhalten Sie von der Floppy die Fehlermeldung RECORD NOT PRESENT. Der Positionier-Befehl wird über den Befehlskanal der Floppy gesendet und hat den folgenden Aufbau:

```
OPEN 15,8,15
PRINT#15,"P"+CHR$(SA)+CHR$(PL)+CHR$(PH)+CHR$(RP)
CLOSE 15
```

bzw. in Assembler:

```
100 ;* befehlskanal oeffnen *
110          lda #$0f      ;logische file-nr. 15
120          ldx #$08      ;geraeteadresse
130          ldy #$0f      ;sekundaeradr. 15
140          jsr $ffba     ;parameter setzen
150          jsr $ffc0     ;open-routine aufrufen
160          ldx #$0f      ;logische file-nr. 15
170          jsr $ffc9     ;datei auf ausgabe schalten
180 ;* positionier-befehl schreiben *
190          ldy #0
200 schleife lda puffer,y  ;zeichen aus puffer holen
210          jsr $ffd2     ;und ausgeben
220          cmp #$0d      ;'return'?
230          beq fertig    ;ja, ausgabe beenden
```

```
240            iny            ;zeiger+1
250            jmp schleife   ;naechstes zeichen
260 fertig     jsr $ffcc      ;auf standardeingabe
                              ;zurückschalten
270 ;* befehlskanal schliessen *
280            lda #$0f       ;logische file-nr. 15
290            jsr $ffc3      ;close-routine aufrufen
300 ;
310 ;* befehlspuffer *
320 puffer     .tx "p"
330            .by sa,pl,ph,rp,13
```

Das ganze sieht schlimmer aus als es ist! SA ist die Sekundäradresse, die Sie beim Öffnen der Datei gewählt haben. PL und PH enthalten die Record-Nummer im sogenannten Low-/High-Byte-Format. PL und PH lassen sich leicht berechnen:

```
PH=INT(RN/256)
PL=RN-256*PH
```

wobei RN die Record-Nummer ist. RP schließlich enthält einen Zeiger auf eine Position innerhalb des Records. Möchte man den gesamten Record schreiben oder lesen, muß man RP auf 1 setzen. Ein Beispiel:

```
.....
200 rn=372:rem record-nummer
210 ph=int(rn/256)
220 pl=rn-256*ph
230 rp=20
200 print#15,"p"+chr$(2)+chr$(pl)+chr$(ph)+chr$(rp)
.....
```

Der PRINT#-Befehl in Zeile 200 setzt den Record-Zeiger auf das 20. Byte des 372. Records. Um es nicht zu vergessen: Nachdem wir mit der Dateiarbeit fertig sind, müssen wir die Datei durch ein

```
CLOSE 2
```

bzw. in Assembler durch:

```
500            lda #$02       ;logische file-nr. 2
510            jsr $ffc3      ;close-routine aufrufen
```

wieder schließen. Die komplette Befehlsfolge zum Einrichten einer relativen Datei sieht also so aus:

```
100 rl=100:rem recordlaenge
110 rn=500:rem anzahl der records
120 ph=int(rn/256)
130 pl=rn-256*ph
140 rp=1:rem position innerhalb record
150 :
160 open 2,8,2,"name,l,"+chr$(rl)
170 open 15,8,15:rem befehlskanal
180 :
190 print#15,"p"+chr$(2)+chr$(pl)+chr$(ph)+chr$(rp)
200 print#2,chr$(255)
210 :
220 close 2
230 close 15
```

Daten schreiben und lesen

Das Schreiben und Lesen eines Records gestaltet sich nun denkbar einfach:

- Der Befehlskanal und die relative Datei werden geöffnet.
- Auf den betreffenden Record wird positioniert.
- Der Record wird geschrieben bzw. gelesen.
- Die Datei und der Befehlskanal werden geschlossen.

Zum Schreiben eines Records verwenden wir den PRINT#-Befehl. Angenommen, der Record ist in RD$ enthalten, dann schreibt ihn ein

```
PRINT#2,RD$
```

in die Datei. Wenn Sie das abschließende "Return" nicht benötigen, schreiben Sie:

```
PRINT#2,RD$;
```

In Assembler sieht das Schreiben eines Datensatzes wie folgt aus. Der Datensatz befindet sich dabei in dem Puffer in Programmzeile 450:

```
320          ldx #$02       ;logische file-nr. 2
330          jsr $ffc9      ;datei auf ausgabe schalten
340 ;
350          ldy #0
360 schleife lda puffer,y   ;zeichen aus puffer holen
370          jsr $ffd2      ;und ausgeben
380          cmp #$0d       ;'return'?
390          beq fertig     ;ja, ausgabe beenden
400          iny            ;zeiger+1
410          jmp schleife   ;naechstes zeichen
420 ;
430 fertig   jsr $ffcc      ;auf standardein-/
                            ;ausgabe zurücksch.
440 ;
450 puffer   .tx "...datensatz..."
460          .by 13
```

Zum Lesen eines Datensatzes stehen die Befehle INPUT# und GET# zur Verfügung. INPUT# benötigt am Ende des Records ein abschließendes "Return". Außerdem kann INPUT# nur Datensätze mit einer Länge bis zu 88 Zeichen einlesen. Mit Hilfe einer GET#-Schleife lassen sich auch längere Records einlesen.

```
FOR Z=1 TO 200:GET#2,EG$:DT$=DT$+EG$:NEXT Z
```

beispielsweise liest einen 200 Byte langen Datensatz ein. In Assembler sieht das Lesen eines Datensatzes wie folgt aus. Der Datensatz wird dabei in den in Programmzeile 450 eingerichteten Puffer eingelesen, der ausreichend groß dimensioniert sein muß:

```
320          ldx #$02       ;logische file-nr. 2
330          jsr $ffc6      ;datei auf eingabe schalten
340 ;
350          ldy #0
360 schleife lda puffer,y   ;zeichen aus puffer holen
370          jsr $ffd2      ;und ausgeben
380          cmp #$0d       ;'return'?
390          beq fertig     ;ja, ausgabe beenden
400          iny            ;zeiger+1
410          jmp schleife   ;naechstes zeichen
420 ;
430 fertig   jsr $ffcc      ;auf standardein-/
                            ;ausgabe zurücksch.
440 ;
450 puffer   .tx "...datensatz..."
```

Da die einzelnen Records in Ihren Programmen häufig länger als 88 Bytes sein werden und die Arbeit mit GET#-Schleifen sehr

zeitaufwendig ist, habe ich die folgende Maschinensprache-Routine DGETV geschrieben.

DGETV liest aus einer beliebigen Floppy-Datei eine anzugebende Anzahl Daten in eine String-Variable ein. Dabei können bis zu 255 Zeichen in einem Stück gelesen werden! Der Aufruf von DGETV erfolgt mit

```
SYS 49737,LF,VR$,LE
```

LF ist die gewohnte logische File-Nummer der Datei, aus der gelesen werden soll. VR$ steht für den Namen der Einlesevariablen. LE schließlich gibt an, wie viele Daten eingelesen werden sollen. Anstelle von

```
FOR Z=1 TO 200:GET#2,EG$:DT$=DT$+EG$:NEXT Z
```

können sie nun schreiben:

```
SYS 49737,2,DT$,200
```

Im Anschluß an das folgende Assembler-Listing finden Sie wieder einen BASIC-Lader zu DGETV:

```
100 ;**********************************************************
105 ;*                                                        *
110 ;* programm: dgetv                                        *
120 ;* liest daten aus einem beliebigen floppyfile            *
130 ;* in eine stringvariable ein                             *
140 ;*                                                        *
150 ;**********************************************************
160 ;*                                                        *
170 ;* aufruf: sys 49737,lf,vr$,le                            *
175 ;* lf: logische file-nummer der datei                     *
180 ;* vr$: name der einlesevariablen                         *
185 ;* le: anzahl der zu lesenden daten (1-255!)              *
190 ;*                                                        *
195 ;**********************************************************
200 ;
205 ;
210 .ba 49737   ;*** startadresse ***
220 ;
230 ;
240 ;*** labels **********************************************
250 .gl zp1 = 251    ;zwischenspeicher
260 .gl zp2 = 252
```

```
265 .gl zp3 = 253
270 ;
280 ;
290 ;*** parameter einlesen **********************************
310          jsr $aefd      ;auf komma testen
320          jsr $b79e      ;logische file-nummer holen
330          jsr $e11e      ;eingabekanal definieren
340          jsr $aefd      ;auf komma testen
350          jsr $b08b      ;variablenname einlesen
360          sta $49        ;variablenadresse zwischensp.
370          sty $4a
380          jsr $b6a3      ;stringparameter holen
390          jsr $aefd      ;auf komma testen
400          jsr $b79e      ;laenge einlesen
410          txa
420          bne dg1        ;<>0, dann weiter
430          jmp $b248      ;'illegal quantity error'
440 dg1      txa            ;laenge in akku
450          jsr $b4f4      ;platz fuer string reservieren
460          sta zp1        ;stringlaenge
470          stx zp2        ;adresse (low)
480          sty zp3        ;adresse (high)
490          ldy #2         ;stringadresse in
500 dg2      lda zp1,y      ;stringdescriptor einschreiben
510          sta ($49),y
520          dey
530          bpl dg2
540 ;
550 ;*** einlese-routine ***********************************
560          ldy #0         ;zaehler initialisieren
570 dg3      jsr $ffcf      ;ein byte von floppy holen
580          sta (zp2),y    ;und ablegen
590          iny
600          cpy zp1        ;eingabelaenge erreicht?
610          bne dg3        ;nein, dann naechstes byte
620          jmp $ffcc      ;kanaele zurücksetzen
```

Und hier der BASIC-Loader:

```
100 rem ******************************************************
105 rem *                                                    *
110 rem * programm: dgetv                                    *
120 rem * programmlaenge: 69 bytes                           *
130 rem * liest daten aus einem beliebigen floppyfile        *
140 rem * in eine stringvariable ein                         *
150 rem *                                                    *
200 rem ******************************************************
205 rem *                                                    *
210 rem * aufruf: sys 49737,lf,vr$,le                        *
220 rem * lf: logische file-nummer der datei                 *
230 rem * vr$: name der einlesevariablen                     *
240 rem * le: anzahl der zu lesenden daten (1-255!)          *
```

```
265 rem *                                                    *
270 rem ******************************************************
280 :
290 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=49737:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 32,253,174,32,158,183,32,30,225,32,1151
1010 data 253,174,32,139,176,133,73,132,74,32,1218
1020 data 163,182,32,253,174,32,158,183,138,208,1523
1030 data 3,76,72,178,138,32,244,180,133,251,1307
1040 data 134,252,132,253,160,2,185,251,0,145,1514
1050 data 73,136,16,248,160,0,32,207,255,145,1272
1060 data 252,200,196,251,208,246,76,204,255,0,1888
2000 data -1:rem endmarkierung
```

Da die Routine ja recht kurz ist, dürfte sie jeder schnell abgetippt haben. Ich werde deshalb im folgenden nur noch mit

```
PRINT#2,RD$;
```

zum Schreiben eines Records (DGETV benötigt ja kein abschließendes "Return") und

```
SYS 49737,2,RD$,RL
```

zum Lesen eines Records arbeiten. Ein Problem, das wir bisher völlig außer acht gelassen haben, ist die Länge der einzelnen Datenfelder. Nehmen wir einmal an, Sie haben eine Adreßdatei mit den Datenfeldern

```
Nachname
Vorname
Strasse
Ort
Telefon
```

Zwei typische Datensätze dieser Datei könnten z.B. so aussehen:

```
Mueller
Klaus
Kastanienweg. 63
8000 Muenchen 60
12 34 56


Schmidhuber
Uwe
Forststr. 178
1000 Berlin 32
56 34 12
```

Wenn Sie die Feldinhalte der beiden Sätze vergleichen, werden Sie sofort feststellen, daß einzig die beiden Telefonnummern dieselbe Länge haben. Alle anderen Feldinhalte weisen eine unterschiedliche Länge auf, wodurch sich natürlich auch verschiedene Gesamtlängen ergeben. Der erste Datensatz ist 52 Byte groß, der zweite 49 Byte.

Bei einer relativen Datei müssen alle Datensätze aber dieselbe Länge haben! Was also tun? Ganz einfach, die Datenfelder werden vor dem Zusammenfassen zu einem Datensatz auf eine bestimmte, vorher festgelegte Länge formatiert.

Wir müssen also zuerst einmal überlegen, wie lang der Inhalt jedes Datenfelds maximal werden kann. Dabei dürfen wir ruhig großzügig sein:

```
Nachname (NN$): 25 Zeichen
Vorname  (VN$): 15 Zeichen
Strasse  (SR$): 20 Zeichen
Ort      (OT$): 20 Zeichen
Telefon  (TF$): 12 Zeichen
```

Damit ergibt sich eine Record-Länge von 92 Byte. Das folgende Programmstück richtet eine passende relative Datei für 50 Datensätze ein:

```
100 rl=92:rem recordlaenge
110 rn=50:rem anzahl der records
120 ph=int(rn/256)
130 pl=rn-256*ph
```

```
140 rp=1:rem position innerhalb record
150 :
160 open 2,8,2,"name,l,"+chr$(rl)
170 open 15,8,15:rem befehlskanal
180 :
190 print#15,"p"+chr$(2)+chr$(pl)+chr$(ph)+chr$(rp)
200 print#2,chr$(255)
210 :
220 close 2
230 close 15
```

Zur Formatierung der Datenfelder verwenden wir die bekannten String-Funktionen LEN, LEFT$, MID$ und RIGHT$. Am Beispiel des Nachnamefeldes möchte ich Ihnen den gesamten Formatiervorgang einmal demonstrieren. Die Maximallänge des Feldes soll ja 25 Byte sein. Zuerst sorgen wir dafür, daß diese Länge nicht überschritten wird:

```
10 nn$=left$(nn$,25)
```

Falls der Nachname jedoch zu kurz ist, muß er mit Leerzeichen aufgefüllt werden. Diese Aufgabe übernimmt eine IF-Abfrage:

```
20 if len(nn$)<25 then nn$=nn$+" ":goto 20
```

Durch diese beiden Anweisungen ist das Nachnamefeld nun auf jeden Fall 25 Byte lang. Beide Anweisungen lassen sich aber auch in einer zusammenfassen:

```
10 lr$="                         ":rem leerstring (25 byte)
20 nn$=left$(nn$+left$(lr$,abs(25-len(nn$))),25)
```

Die Anweisung in Zeile 20 sieht komplizierter aus als sie ist. Falls NN$ kürzer als 25 Zeichen ist, wird es durch das innere LEFT$ um die entsprechende Anzahl Leerzeichen aus LR$ aufgefüllt. Ist NN$ dagegen länger als 25 Zeichen, werden die überzähligen Zeichen durch das äußere LEFT$ abgeschnitten.

Die Funktion ABS sorgt in diesem Fall dafür, daß beim inneren LEFT$ kein negativer Wert auftritt. Die anderen Felder lassen sich ebenso formatieren. Damit können wir uns ein Programm zum Eingeben und Schreiben eines Datensatzes erstellen:

```
100 rem datei oeffnen
105 rl=92:rem recordlaenge
110 open 2,8,2,"name,l,"+chr$(rl)
120 open 15,8,15:rem befehlskanal
130 :
135 rem adresse eingeben
140 input "nachname: ";nn$
150 input "vorname: ";vn$
160 input "strasse: ";sr$
170 input "ort: ";ot$
180 input "telefon: ";tf$
185 :
190 rem felder formatieren
200 lr$="                         ":rem leerstring (25 byte)
210 nn$=left$(nn$+left$(lr$,abs(25-len(nn$))),25)
220 vn$=left$(vn$+left$(lr$,abs(15-len(vn$))),15)
230 sr$=left$(sr$+left$(lr$,abs(20-len(sr$))),20)
240 ot$=left$(ot$+left$(lr$,abs(20-len(ot$))),20)
250 tf$=left$(tf$+left$(lr$,abs(12-len(tf$))),12)
255 :
260 rem felder zu einem record zusammenfassen
270 rd$=nn$+vn$+sr$+ot$+tf$
275 :
280 rem gewuenschte record-nummer
290 print:input "record-nummer: ";rn
295 :
300 rem auf record positionieren
310 ph=int(rn/256)
320 pl=rn-256*ph
330 rp=1
340 print#15,"p"+chr$(2)+chr$(pl)+chr$(ph)+chr$(rp)
345 :
350 rem record schreiben
360 print#2,rd$;
370 :
380 rem noch eine adresse?
390 input "moechten sie noch eine adresse eingeben:";jn$
400 if jn$="j" then 140
410 :
420 rem datei schliessen
430 close 2
440 close 15
```

Geben Sie mit dem Programm nun einmal mehrere Adressen ein. Die Record-Nummern dürfen Sie zwischen 1 und 50 frei wählen. Mit Hilfe des nun folgenden Programmstücks können Sie sich die Adressen dann wieder ausgeben lassen:

```
500 rem datei oeffnen
510 rl=92:rem recordlaenge
520 open 2,8,2,"name,l,"+chr$(rl)
```

```
530 open 15,8,15:rem befehlskanal
540 :
550 rem gewuenschte record-nummer
560 input "record-nummer: ";rn
575 :
580 rem auf record positionieren
590 ph=int(rn/256)
600 pl=rn-256*ph
610 rp=1
620 print#15,"p"+chr$(2)+chr$(pl)+chr$(ph)+chr$(rp)
635 :
640 rem record lesen
650 sys 49737,2,rd$,rl
660 :
670 rem record in felder aufteilen
680 nn$=mid$(rd$,1,25)
690 vn$=mid$(rd$,26,15)
700 sr$=mid$(rd$,41,20)
710 ot$=mid$(rd$,61,20)
720 tf$=mid$(rd$,81,12)
730 :
740 rem adresse ausgeben
750 print "nachname: ";nn$
760 print "vorname: ";vn$
770 print "strasse: ";sr$
780 print "ort: ";ot$
790 print "telefon: ";tf$
800 :
810 rem noch eine adresse?
820 input "noch eine adresse ausgeben:";jn$
830 if jn$="j" then 560
840 :
850 rem datei schliessen
860 close 2
870 close 15
```

Achtung: Bitte achten Sie darauf, daß sich die Routine DGETV beim Start des Programms im Rechnerspeicher befindet! Ansonsten bekommen sie in der Programmzeile 650 einen Systemabsturz.

Von besonderem Interesse sind die Programmzeilen 680 bis 720. Dort wird der gelesene Record mit MID$ wieder in die einzelnen Datenfelder aufgeteilt.

5.4 Indexsequentielle Dateien

Ein nicht zu unterschätzender Nachteil der relativen Datenspeicherung ist die fehlende Möglichkeit, auf einen bestimmten Datensatz direkt über einen Feldinhalt zugreifen zu können, bei einer Adreßdatei also beispielsweise die zu dem Nachnamen "Mueller" gehörende Adresse direkt laden zu können.

Statt dessen muß man die Record-Nummer des betreffenden Datensatzes kennen. Eine mögliche Abhilfe wäre es, die Datensätze ständig alphabetisch sortiert zu halten. Bei großen Datenmengen ist der Aufwand dazu aber viel zu groß.

Deshalb bedient man sich häufig einer anderen Methode, der sogenannten "indexsequentiellen Datei". Die Grundidee ist dabei die folgende: Die eigentlichen Daten, beispielsweise Adressen, werden in einer relativen Datei abgelegt. Zusätzlich wird eine separate sequentielle Datei, eine sogenannte "Indexdatei", angelegt, in der die Inhalte eines Datenfeldes sowie die zugehörigen Record-Nummern der entsprechenden Datensätze der relativen Datei gespeichert werden.

Bei einer Adreßdatei nimmt man dazu in der Regel das Nachnamefeld, über das man ja am häufigsten auf eine Adresse zugreift. Natürlich läßt sich aber jedes beliebige Feld verwenden. Sogar mehrere sequentielle Dateien sind denkbar. Bei einer Rechnungsdatei z.B. könnte man sowohl die Rechnungsnummern als auch die Namen der Kunden als Index verwenden.

Möchte man nun mit der Datei arbeiten, so wird zu Beginn die komplette Indexdatei in den Rechner geladen. Da sie ja nur aus den Inhalten eines Datenfeldes und den Record-Nummern besteht, nimmt sie nicht allzu viel Speicherplatz in Anspruch.

Gibt man jetzt einen Nachnamen ein, dann sucht die Dateiverwaltung den Nachnamen in der Indexdatei (was natürlich blitzschnell geht), holt sich die zugehörige Record-Nummer und lädt damit den Record der relativen Datei, der die entsprechende Adresse enthält.

Das nun folgende Programm ist ein Beispiel für eine Adreßverwaltung mit indexsequentieller Dateiorganisation. Das Programm ist bewußt recht schlicht gehalten. Ich möchte Ihnen hier ja nicht nur fertige Lösungen anbieten, sondern Sie auch zu eigener Programmiertätigkeit animieren. Und mit den Kenntnissen aus den letzten beiden Abschnitten dürfte es Ihnen nicht schwer fallen, das Programm individuell zu ergänzen.

Beginnen wir mit dem Programmkopf und den Vorbereitungen. Die Kommentarzeilen müssen Sie natürlich nicht alle mit abtippen. Ganz weglassen sollten Sie sie allerdings auch nicht. Sonst verlieren Sie später leicht die Übersicht:

```
100 rem ***************************************************
110 rem *                                                 *
120 rem * programm: adressen                              *
130 rem * adreßverwaltung mit indexsequentieller          *
140 rem * dateiorganisation                               *
150 rem *                                                 *
160 rem ***************************************************
170 :
180 :
190 :
200 rem *** initialisierung *******************************
210 :
220 rem maschinenprogramm nachladen
230 if f=0 then f=1:load "dgetv",8,1
240 dm=250:rem maximale datensatzanzahl
250 dim id$(dm):rem indexfeld (nachnamen)
260 dim in(dm):rem zugehörige record-nummern
270 dim ds$(7):rem datenfelder
280 gosub 510:rem datei oeffnen
```

Die Maximalanzahl der Adressen wird in Zeile 240 auf 250 festgesetzt. Diesen Wert können Sie natürlich nach Belieben abändern. Die Variable DM wird im gesamten Programm verwendet, so daß diese eine Änderung in Zeile 240 genügt.

```
290 :
300 :
310 :
320 rem *** auswahlmenue ***********************************
330 :
340 wl=0:print:print
350 print tab(11) "adressenverwaltung"
360 print
370 print tab(8) "-1-: adressen eingeben"
```

```
380 print tab(8) "-2-: adressen ausgeben"
390 print tab(8) "-5-: ende"
400 print
410 print tab(11) "ihre wahl (1,2,5)";:input wl
420 if wl=1 then gosub 640:rem adressen eingeben
430 if wl=2 then gosub 970:rem adressen ausgeben
440 if wl=5 then goto 1310:rem ende
450 goto 340:rem zum auswahlmenue
460 :
470 :
480 :
```

Das Auswahlmenü des Programms besteht aus den drei Menüpunkten

```
ADRESSEN EINGEBEN
ADRESSEN AUSGEBEN
ENDE
```

Wenn Sie das Programm um neue Funktionen erweitern wollen, etwa

```
ADRESSEN LOESCHEN oder ADRESSEN AENDERN
```

gehen Sie wie folgt vor: Nachdem Sie für die neue Programmfunktion ein entsprechendes Unterprogramm geschrieben haben, fügen Sie den neuen Menüpunkt zwischen den Programmzeilen 390 und 400 ein und weisen ihm eine Kennziffer zu.

Anschließend erweitern Sie die IF-Abfragen in den Zeilen 420 bis 440 um eine weitere Abfrage, die in Ihr Unterprogramm verzweigt. Das ist schon alles!

```
490 rem *** datei oeffnen ************************************
500 :
510 lf=1:sa=2:dn$="adressen":rl=107:gosub 1490:rem relative datei
520 open 2,8,3,"index,s,r":rem sequentielle datei
530 gosub 1420:rem floppyfehlerkanal auslesen
540 if er<>0 then ad=0:goto 570:rem datei noch nicht angelegt
550 input#2,ad:rem datensatzanzahl
560 for i=1 to ad:input#2,id$(i),in(i):next i:rem indexdatei
570 close 2:rem sequentielle datei
580 return
590 :
600 :
610 :
```

Das erste Unterprogramm öffnet die relative und die sequentielle Datei und liest die Indexdaten ein. Die relative Datei hat den Namen "ADRESSEN". Die Record-Länge beträgt 107 Bytes. Die sequentielle Datei trägt den Namen "INDEX". In ihr sind die Nachnamen gespeichert.

Da die Nachnamen in derselben Reihenfolge wie die Adressen in der relativen Datei abgelegt sind, ist die jeweilige Record-Nummer mit der Position des Nachnamens innerhalb der Indexdatei identisch. Die Adresse des 10. Nachnamens beispielsweise befindet sich im Record 10. Daher müssen die Record-Nummern nicht extra gespeichert werden.

```
620 rem *** adressen eingeben ********************************
630 :
640 print:print
650 input "nachname (15) :";ds$(1)
660 input "vorname (10) :";ds$(2)
670 input "strasse (15) :";ds$(3)
680 input "wohnort (15) :";ds$(4)
690 input "telefon (11) :";ds$(5)
700 input "bemerkung1 (20) :";ds$(6)
710 input "bemerkung2 (20) :";ds$(7)
720 ad=ad+1:rem anzahl der adressen
730 id$(ad)=ds$(1):rem nachname in indexdatei
740 in(ad)=rn:rem record-nummer
750 rem eingabefelder auf richtige laenge bringen
760 le$="                    ":rem zum auffuellen der felder
770 ds$(1)=left$(ds$(1)+left$(le$,abs(15-len(ds$(1)))),15)
780 ds$(2)=left$(ds$(2)+left$(le$,abs(10-len(ds$(2)))),10)
790 ds$(3)=left$(ds$(3)+left$(le$,abs(15-len(ds$(3)))),15)
800 ds$(4)=left$(ds$(4)+left$(le$,abs(15-len(ds$(4)))),15)
810 ds$(5)=left$(ds$(5)+left$(le$,abs(11-len(ds$(5)))),11)
820 ds$(6)=left$(ds$(6)+left$(le$,abs(20-len(ds$(6)))),20)
830 ds$(7)=left$(ds$(7)+left$(le$,abs(20-len(ds$(7)))),20)
840 rem datensatz zum speichern zusammenfassen
850 rc$="":for i=1 to 7:rc$=rc$+ds$(i):next i
860 rem datensatz speichern
870 rn=ad:rp=1:gosub 1570:rem auf record positionieren
880 print#lf,rc$:rem datensatz in record schreiben
890 gosub 1420:rem floppyfehlerkanal auslesen
900 print:print:print tab(5) "adresse ist gespeichert!"
910 return
920 :
930 :
940 :
```

Für die Adreßeingabe stehen Ihnen sieben Datenfelder zur Verfügung:

```
NACHNAME
VORNAME
STRASSE
WOHNORT
TELEFON
BEMERKUNG1
BEMERKUNG2
```

Die Zahlen in Klamern hinter den Datenfeldbezeichnungen geben die maximale Datenfeldlänge an. Für den Nachnamen stehen Ihnen also 15 Zeichen zur Verfügung, für den Vornamen 10 Zeichen usw...

Wenn Sie diese Werte überschreiten, ist das aber auch nicht weiter schlimm. Die einzelnen Felder werden in den Programmzeilen 770 bis 830 automatisch auf die richtige Länge formatiert.

```
950 rem *** adressen ausgeben ******************************
960 :
970 print:print:input "nachname";nn$
980 for i=1 to ad:rem index suchen
990 if nn$=id$(i) then gn=i:i=ad:next i:goto 1040
1000 next i
1010 print:print:print tab(5) "adresse nicht vorhanden!"
1020 return
1030 rem adresse gefunden
1040 rn=gn:rp=1:gosub 1570:rem auf record positionieren
1050 sys 49737,lf,rc$,107:record einlesen
1060 rem record aufteilen
1070 ds$(1)=mid$(rc$,1,15):rem nachname
1080 ds$(2)=mid$(rc$,16,10):rem vorname
1090 ds$(3)=mid$(rc$,26,15):rem strasse
1100 ds$(4)=mid$(rc$,41,15):rem wohnort
1110 ds$(5)=mid$(rc$,56,11):rem telefon
1120 ds$(6)=mid$(rc$,67,20):rem bemerkung1
1130 ds$(7)=mid$(rc$,87,20):rem bemerkung2
1140 rem datensatz ausgeben
1150 print:print
1160 print "nachname: ";ds$(1)
1170 print "vorname: ";ds$(2)
1180 print "strasse: ";ds$(3)
1190 print "wohnort: ";ds$(4)
1200 print "telefon: ";ds$(5)
1210 print "bemerkung1: ";ds$(6)
1220 print "bemerkung2: ";ds$(7)
1230 print:print:print "weiter mit 'space'!"
```

```
1240 get eg$:if eg$<>chr$(32) then 1240
1250 return
1260 :
1270 :
1280 :
```

Möchten Sie sich nun eine bestimmte Adresse ausgeben lassen, fragt Sie das Programm nach dem Nachnamen und sucht diesen dann in dem Variablenfeld ID$. War die Suche erfolgreich, dann ist der Schleifenzähler I gleich der gesuchten Record-Nummer. Der betreffende Record wird mit der Routine DGETV aus dem letzten Abschnitt an einem Stück eingelesen, mit MID$ in die einzelnen Datenfelder aufgeteilt und zusammen mit den entsprechenden Feldbezeichnungen am Bildschirm ausgegeben.

```
1290 rem *** ende *********************************************
1300 :
1310 print#15,"s:index":rem indexdatei loeschen
1320 open 2,8,3,"index,s,w":rem und neu schreiben
1330 print#2,ad:rem datensatzanzahl
1340 for i=1 to ad:print#2,id$(i)",","in(i):next i
1350 gosub 1650:rem relative datei schliessen
1360 end
1370 :
1380 :
1390 :
```

Sobald Sie das Programm beenden, wird die Indexdatei komplett auf die Diskette zurückgeschrieben und alle Dateien geschlossen. Die folgenden vier Unterprogramme lassen sich in jedem Programm, das mit relativen Dateien arbeitet, gut einsetzen:

```
1400 rem *** floppyfehlerkanal auslesen *******************
1410 :
1420 input#15,er,er$,tr,sk
1430 return
1440 :
1450 :
1460 :
1470 rem *** relative datei oeffnen ***********************
1480 :
1490 open 15,8,15,:rem floppyfehlerkanal
1500 open lf,8,sa,dn$+",l,"+chr$(rl):rem datei
1510 return
1520 :
1530 :
1540 :
1550 rem *** auf record positionieren *********************
```

```
1560 :
1570 hb=int(rn/256):lb=rn-256*hb
1580 print#15,"p"+chr$(sa)+chr$(lb)+chr$(hb)+chr$(rp)
1590 return
1600 :
1610 :
1620 :
1630 rem *** relative datei schliessen ********************
1640 :
1650 close lf:rem datei
1660 close 15:rem floppyfehlerkanal
1670 return
```

5.5 User- und Programmdateien

Was Programmdateien sind ist Ihnen mittlerweile ja bestens bekannt. Daher möchte ich die wichtigsten Befehle hier nur noch einmal kurz wiederholen. Es kommt aber auch einiges Neue hinzu.

Grundsätzlich kann man zwischen zwei Arten von Programmen unterscheiden: BASIC- und Maschinensprache-Programme. Beide Arten werden im Directory durch das Kürzel "PRG" gekennzeichnet. Ein BASIC-Programm läßt sich mit

```
SAVE "NAME",8
```

auf Diskette abspeichern. Möchte man sichergehen, daß das Programm auch korrekt gespeichert wurde, so kann man es anschließend mit

```
VERIFY "NAME",8
```

überprüfen lassen. VERIFY vergleicht das im Speicher befindliche BASIC-Programm Byte für Byte mit dem auf der Diskette abgelegten. Stimmen beide Progamme überein (was die Regel ist), erhalten Sie die Meldung

```
OK
```

ansonsten einen

```
VERIFY ERROR
```

Falls Sie versuchen, ein Programm mit einem Namen zu speichern, unter dem bereits eine andere Datei auf der Diskette abgelegt ist, reagiert die Floppy mit einem FILE EXISTS-Fehler. Es würde ja auch keinen Sinn machen, zwei Dateien mit denselben Namen auf einer Diskette zu haben. In diesem Fall müssen Sie entweder einen anderen Namen wählen oder die alte Datei mit

```
OPEN 15,8,15,"S:NAME":CLOSE 15
```

löschen. Im Handbuch zur Floppy ist auch noch eine andere Möglichkeit angegeben:

```
SAVE"@:NAME",8
```

Durch den vorangestellten "Klammeraffen" im SAVE-Befehl wird die alte Datei ebenfalls gelöscht.

Warnung: Bitte vergessen Sie diese Möglichkeit so schnell wie möglich, und wenden Sie sie niemals an! Infolge eines Fehlers im Betriebssystem der Floppy wird durch den Klammeraffen unter Umständen der gesamte Disketteninhalt zerstört!

Wieder in den Rechnerspeicher geladen wird ein BASIC-Programm mit

```
LOAD "NAME",8
```

Anschließend kann es mit RUN gestartet oder mit LIST am Bildschirm gelistet werden. Wie ich Ihnen bereits geraten habe, ist es am zweckmäßigsten, wenn man häufig benötigte Programmteile (beispielsweise Eingabe- oder Ausgaberoutinen) als Unterprogramme schreibt und in separaten Programmdateien ablegt.

Benötigt man dann später eines dieser Unterprogramme, hängt man es einfach an das betreffende Programm hinten an. Den Vorgang des "Anhängens" bezeichnet man auch als "Mergen". Wie

geht man dazu nun konkret vor? Schließlich kann man mit LOAD ja immer nur ein Programm in den Rechner holen.

Ein sehr wichtiger Punkt vornweg: Damit das zusammengefügte Programm nachher überhaupt lauffähig ist, müssen die beiden Einzelprogramme über unterschiedliche Zeilennummern verfügen. Außerdem müssen die Zeilennummern des zweiten (angehängten) Programms höher sein als die des ersten.

Sie können also beispielsweise keine zwei Programme mergen, die beide mit der Programmzeile 100 beginnen. Das sollten Sie beim Aufbau Ihrer Unterprogrammsammlung von vornherein beachten. Am einfachsten ist es, wenn Sie jedes Unterprogramm mit einer neuen 1000er-Nummer beginnen, beispielsweise

```
10000 rem unterprogramm 1
.....
10990 return
11000 rem unterprogramm 2
.....
11990 return
12000 rem unterprogramm 3
.....
12990 return
usw.
```

Wenn Sie mehrere Unterprogramme anhängen wollen, müssen Sie auf die Reihenfolge achten, also beispielsweise nicht zuerst das Unterprogramm 3 und dann das Unterprogramm 1 mergen, da ja sonst die Reihenfolge der Zeilennummern nicht stimmen würde.

Doch kommen wir zum Mergen selbst. Laden Sie dazu das erste Programm in den Rechner. Wie Sie ja wissen, ist der gesamte Speicher des Commodore 64 in einzelne Speicherzellen unterteilt, die von Null bis 65.535 "durchnumeriert" sind.

Ein BASIC-Programm, das in den Rechnerspeicher geladen wird, wird nun in einem bestimmten Speicherbereich abgelegt. Den Anfang und das Ende dieses BASIC-Speichers merkt sich der Rechner in vier Speicherzellen:

```
43/44: Programmbeginn
45/46: Programmende
```

Zuerst lesen wir mit

```
PRINT PEEK(43),PEEK(44)
```

die Inhalte der Speicherzellen 43 und 44 aus und merken sie uns (nicht in Variablen, da deren Inhalte beim Mergen zerstört werden!). Anschließend ermitteln wir mit

```
ED=PEEK(45)+PEEK(46)*256
```

das Programmende, vermindern diesen Wert mit

```
ED=ED-2
```

um 2, teilen das Ergebnis wieder in ein Low- und ein High-Byte auf

```
HB=INT(ED/256):LB=ED-HB*256
```

und setzen diesen Wert als neuen Programmanfang

```
POKE 43,LB:POKE 44,HB
```

Jetzt können wir mit

```
LOAD "NAME",8
```

das zweite Programm nachladen. Es wird nun direkt "hinter" das erste Programm geladen. Zum Schluß setzen wir wieder den alten Programmanfang. Die beiden Werte haben Sie sich ja gemerkt. Normalerweise müssen Sie das aber nicht einmal, da die Werte in der Regel immer dieselben sind, nämlich 1 und 8:

```
POKE 43,1:POKE 44,8
```

Das war schon alles! Das neu entstandene Programm können Sie nun mit SAVE wie gewohnt abspeichern. Das Mergen läßt sich übrigens beliebig oft wiederholen; natürlich nur so lange, bis der Rechnerspeicher voll ist. Sie können also ohne weiteres mehrere Programme aneinanderhängen, ohne die Zwischenprodukte extra abspeichern zu müssen.

Das Speichern und Laden von Maschinenprogrammen gestaltet sich aber leider nicht ganz so einfach.

Ich habe Ihnen ja schon zwei Routinen vorgestellt, mit denen Sie Maschinenprogramme sehr komfortabel abspeichern und an eine beliebge Speicheradresse wieder einladen können. Und auch der "normale" LOAD-Befehl läßt sich zum Laden von Maschinenprogrammen einsetzen:

```
LOAD "NAME",8,1
```

lädt ein Maschinenprogramm an die Adresse, von der aus es gespeichert würde. Dieser Befehl hat allerdings seine Tücken. Er verstellt nämlich den Zeiger auf den BASIC-Programmanfang (den ich Ihnen ja gerade beim Mergen vorgestellt habe) auf den Beginn der Maschinenroutine. Nach einem solchen LOAD-Befehl ist daher ein NEW erforderlich, um den Zeiger wieder "zurechtzubiegen".

Das gilt allerdings nur, wenn man im Direktmodus arbeitet. Im Programmodus gibt es aber ein anderes Problem: Nach einem LOAD-Befehl beginnt das BASIC-Programm immer wieder von vorne! Möchten Sie also eine Maschinenroutine nachladen und schreiben deshalb:

```
100 rem maschinenroutine nachladen
110 load "name",8,1
120 rem weiteres programm
```

so geraten Sie in eine Endlosschleife. Eine IF-Abfrage schafft dem Abhilfe:

```
100 rem maschinenroutine nachladen
110 if fl=0 then fl=1:load "name",8,1
120 rem weiteres programm
```

Sobald das Maschinenprogramm nachgeladen ist und das BASIC-Programm ein zweites Mal in Zeile 110 beginnt, hat die Variable FL ja den Wert 1. Das Programm wird daher in Zeile 120 fortgesetzt. Auch das Einladen mehrerer Maschinenroutinen läßt sich so leicht realisieren:

```
100 rem maschinenroutine1 nachladen
110 if fl=0 then fl=1:load "name1",8,1
130 rem maschinenroutine2 nachladen
140 if fl=1 then fl=2:load "name2",8,1
150 rem maschinenroutine3 nachladen
160 if fl=2 then fl=3:load "name3",8,1
170 rem weiteres programm
```

Es gibt aber auch noch einen ganz anderen Weg, auf Programme (egal ob BASIC oder Maschinensprache) zuzugreifen. Dazu muß man wissen, daß sich ein Programm, wie eine sequentielle oder relative Datei, mit OPEN öffnen läßt:

```
OPEN 2,8,2,"NAME,P,Modus"
```

Das "P" hinter dem Namen signalisiert der Floppy, daß es sich um eine Programmdatei handelt. Mit dem "Modus" müssen Sie wieder festlegen, ob Sie auf die Datei schreibend oder lesend zugreifen wollen.

Diese Art des Zugriffs eröffnet vielfältige Möglichkeiten. So könnte man beispielsweise ein BASIC-Programm Byte für Byte einlesen lassen, um es zu analysieren oder zu verändern. Dazu muß man natürlich die interne Struktur eines BASIC-Programms kennen. Ein anderer Anwendungsfall ist die Ermittlung der Startadresse eines Maschinenprogramms. Diese wird nämlich als die ersten beiden Bytes der Programmdatei gespeichert und läßt sich deshalb leicht lesen:

```
100 open 2,8,2,"name,p,r":rem datei zum lesen oeffnen
110 get#2,lb$:lb=asc(lb$):rem low-byte
120 get#2,hb$:hb=asc(hb$):rem high-byte
130 close 2
140 sa=hb*256+lb:rem startadresse
```

Zum Schluß möchte ich Ihnen noch kurz einen Dateityp vorstellen, den man nur sehr selten benötigt, die sogenannte "User-Datei". Eine User-Datei ist im Grunde genommen nicht anderes als eine sequentielle Datei. Der einzige Unterschied besteht in dem Kürzel USR, mit dem dieser Dateityp im Directory gekennzeichnet ist. Geöffnet wird eine User-Datei mit

```
OPEN 2,8,2,"NAME,U,Modus"
```

Das "U" hinter dem Namen signalisiert der Floppy, daß es sich um eine User-Datei handelt. Mit dem Modus legen Sie wieder fest, ob Sie in die Datei schreiben oder aus ihr lesen wollen. Ansonsten gilt für die User-Dateien dasselbe wie für die sequentiellen Dateien.

Ein Anwendungsfall für eine User-Datei ist beispielsweise das "Listen" eines BASIC-Programms auf Diskette. Möchte man ein BASIC-Programm zum Beispiel mit einer Textverarbeitung einlesen, so darf man dazu nicht das mit SAVE auf Diskette abgelegte Programm verwenden, da das Programm in einem speziellen Format abgespeichert wird, das in einer Textverarbeitung nur ein heilloses Durcheinander, wenn nicht gar einen Programmabsturz erzeugen würde.

Statt dessen erzeugt man eine User-Datei, in die das betreffende Programm gelistet wird. Zuerst öffnet man die User-Datei:

```
OPEN 2,8,2,"NAME,U,W"
```

Anschließend leitet man die Bildschirmausgabe mit

```
CMD 2
```

auf die Datei um und listet dann das Programm wie gewohnt mit

```
LIST
```

Abschließend wird die Datei wieder geschlossen:

```
PRINT#2
CLOSE 2
```

5.6 Der Direktzugriff auf die Diskette

Alle auf einer Diskette gespeicherten Daten sind in Blökken zu je 256 Bytes organisiert. Mit Hilfe einer Direktzugriffsdatei ist es nun möglich, auf jeden einzelnen dieser Blöcke direkt zuzugreifen, d.h. ihn zu lesen und zu schreiben.

Damit lassen sich ohne große Umstände eigene Dateistrukturen verwirklichen. Viel interessanter aber dürfte es sein, Manipulationen an vorhandenen Daten vorzunehmen, z.B. im Directory einer Diskette.

Wie Sie sich leicht denken können, muß man dabei natürlich sehr vorsichtig sein. Selbst kleinste Fehler können den größten Schaden anrichten! So genügt beispielsweise schon ein falsch zurückgeschriebener Directory-Block, um das gesamte Inhaltsverzeichnis einer Diskette durcheinanderzubringen.

Für Ihre ersten Experimente mit dem Direktzugriff sollten Sie nur Kopien Ihrer Programm- und Datendisketten verwenden!

Die logische Abfolge der einzelnen Schritte ist bei einer Direktzugriffsdatei in etwa dieselbe wie bei den anderen Dateien. Wohlgemerkt, nur in etwa. Es gibt einen sehr wichtigen Unterschied: die Zwischenspeicherung der Datenblöcke in einem Pufferspeicher der Floppy. Gehen wir die einzelnen Schritte einmal der Reihe nach durch:

Eine Direktzugriffsdatei öffnen und schließen

Zunächst müssen wir die Direktzugriffsdatei öffnen:

```
OPEN 2,8,2,"#"
```

Damit die Floppy weiß, daß wir eine Direktzugriffsdatei öffnen wollen, verwenden wir als Dateinamen das Doppelkreuz #. Die restlichen drei Parameter kennen Sie ja zur Genüge. Zusätzlich müssen wir auch den Befehlskanal öffnen. Warum, das werden Sie gleich sehen:

```
OPEN 15,8,15
```

In Assembler sieht das komplett so aus:

```
130 ;* befehlskanal oeffnen *
140        lda #$0f      ;logische file-nr. 15
150        ldx #$08      ;geraeteadresse
160        ldy #$0f      ;sekundaeradr. 15
170        jsr $ffba     ;parameter setzen
```

```
175            jsr $ffc0       ;open-routine aufrufen
180 ;
185 ;* direktdatei oeffnen *
190            lda #$02        ;logische file-nr. 2
200            ldx #$08        ;geraeteadresse
210            ldy #$02        ;sekundaeradr. 2
220            jsr $ffba       ;parameter setzen
230            lda #$01        ;textlaenge!
240            ldx #<(text)    ;zeiger auf
250            ldy #>(text)    ;text
260            jsr $ffbd       ;parameter setzen
270            jsr $ffc0       ;open-routine aufrufen
1000 ;
1010 text       .tx "#"         ;dateiname
```

Nachdem wir fertig sind, dürfen wir nicht vergessen, die Dateien wieder zu schließen. Dafür ist, wie gewohnt, der CLOSE-Befehl zuständig:

```
CLOSE 2:CLOSE 15
```

bzw. in Assembler:

```
920            jsr $ffcc       ;auf standardeingabe
                               ;zurückschalten
930            lda #$02        ;direktdatei
940            jsr $ffc3       ;close-routine aufrufen
950            lda #$0f        ;befehlskanal
960            jsr $ffc3       ;close-routine aufrufen
```

Einen Datenblock lesen

Ans Schließen der Datei wollen wir im Moment natürlich noch nicht denken, sondern erst einmal sehen wie man einen Datenblock liest. Wie schon ganz zu Beginn dieses Kapitels kurz erwähnt, sind die Datenblocks auf der Diskette in konzentrischen Spuren (engl.: Tracks) angeordnet. Jede Spur ist in einzelne Sektoren unterteilt, in die jeweils gerade ein Datenblock paßt.

Da der Spurumfang ja nach außen hin immer größer wird, bzw. umgekehrt nach innen immer kleiner, paßt in die einzelnen Spuren nicht dieselbe Anzahl Sektoren. Auf den äußeren Spuren können einige Sektoren mehr untergebracht werden als auf den inneren. Insgesamt enthält jede Diskette 36 Spuren mit der folgenden Sektoreinteilung und Nummernzuordnung:

Spur	Sektor
00 - 17	00-20
18 - 24	00-18
25 - 30	00-17
31 - 35	00-16

Die Spuren sind also von Null bis 35 durchnumeriert. Auf den Spuren 0 bis 17 finden 21 Sektoren Platz, während es auf den Spuren 31 bis 35 nur noch 17 Sektoren sind.

Um einen bestimmten Datenblock anzusprechen, gibt man nun einfach seine Spur- und Sektornummer an. 27/5 beispielsweise bezeichnet den Datenblock auf Spur 27, Sektor 5. Wie teilt man diese beiden Werte nun aber der Floppy mit? Dazu gibt es den "Block-Read-Befehl", der über den Befehlskanal gesendet werden muß (daher haben wir ihn eingangs gleich mit geöffnet). Der Block-Read-Befehl hat das folgende Format:

```
PRINT#15,"U1 2 0";SP;SK
```

bzw. in Assembler:

```
320           ldx #$0f      ;logische file-nr. 15
330           jsr $ffc9     ;datei auf ausgabe schalten
340           ldy #0
350 schleife  lda puffer,y  ;zeichen aus puffer holen
360           jsr $ffd2     ;und ausgeben
370           cmp #$0d      ;'return'?
375           beq fertig    ;ja, ausgabe beenden
380           iny           ;zeiger+1
385           jmp schleife  ;naechstes zeichen
390 fertig    nop           ;weiteres programm
395 ;
1100 puffer   .tx "u1 2 0"
1110          .by sp,sk,13
```

"U1" ist der eigentliche Block-Read-Befehl. Die 2 dahinter ist die Sekundäradresse, die wir beim Öffnen der Direktzugriffsdatei gewählt haben. Die 0 dahinter muß immer angegeben werden. Zum Schluß folgen schließlich die Spurnummer "SP" und die Sektornummer "SK" des anzusprechenden Datenblocks.

Möchten Sie beispielsweise den Datenblock auf Spur 18, Sektor 0 (das ist der erste Directory-Block) laden, so geben Sie ein:

```
PRINT#15,"U1 2 0";18;0
```

Genauso gut können sie auch schreiben:

```
PRINT#15,"U1 2 0 18 0"
```

Durch diese Anweisung wird der Datenblock nun aber nicht in den Rechnerspeicher geholt (wohin auch?), sondern nur in einen internen Pufferspeicher der Floppy geladen. Von dort aus können wir über die Direktzugriffsdatei, die wir mit der logischen File-Nummer 2 geöffnet haben, die Daten Byte für Byte lesen.

Zuvor müssen wir der Floppy aber noch mitteilen, ab welcher Position im Datenblock wir Daten lesen (oder auch schreiben) wollen. Dazu gibt es den "Puffer-Pointer-Befehl", der einen Zeiger auf die gewünschte Position setzt:

```
PRINT#15,"B-P 2";PS
```

bzw. in Assembler:

```
320           ldx #$0f       ;logische file-nr. 15
330           jsr $ffc9      ;datei auf ausgabe schalten
340           ldy #0
350 schleife  lda puffer,y   ;zeichen aus puffer holen
360           jsr $ffd2      ;und ausgeben
370           cmp #$0d       ;'return'?
375           beq fertig     ;ja, ausgabe beenden
380           iny            ;zeiger+1
385           jmp schleife   ;naechstes zeichen
390 fertig    nop            ;weiteres programm
395 ;
1100 puffer   .tx "b-p 2"
1110          .by ps,13
```

Die 2 hinter "B-P" ist wieder die Sekundäradresse. PS enthält den Wert für den Zeiger. Da ein Datenblock ja gerade 256 Byte lang ist, kann man den Zeiger auf eine Position zwischen Null (1. Daten-Byte) und 255 (letztes Daten-Byte) setzen.

```
PRINT#15,"B-P 2";175
```

setzt beispielsweise den Zeiger auf die Position 175. Jetzt endlich können wir die Daten lesen. Dazu benötigen wir den GET#-Befehl.

```
FOR Z=1 TO 6:GET#2,EG$:DT$=DT$+EG$:NEXT Z
```

liest beispielsweise die Daten-Bytes 175 bis 180 in DT$ ein.

Einen Datenblock schreiben

Analog dazu können Sie mit PRINT# in den Datenblock schreiben. Geben Sie beispielsweise ein:

```
PRINT#15,"B-P 2";175
PRINT#2,"BEISPIEL";
```

so steht der Text "BEISPIEL" anschließend im Puffer von Position 175 bis 182. Um den gesamten Datenblock wieder auf die Diskette zurückzuschreiben, bedarf es des "Block-Write-Befehls", der wie folgt aufgebaut ist:

```
PRINT#15,"U2 2 0";SP;SK
```

bzw. in Assembler:

```
320          ldx #$0f       ;logische file-nr. 15
330          jsr $ffc9      ;datei auf ausgabe schalten
340          ldy #0
350 schleife lda puffer,y   ;zeichen aus puffer holen
360          jsr $ffd2      ;und ausgeben
370          cmp #$0d       ;'return'?
375          beq fertig     ;ja, ausgabe beenden
380          iny            ;zeiger+1
385          jmp schleife   ;naechstes zeichen
390 fertig   nop            ;weiteres programm
395 ;
1100 puffer  .tx "u2 2 0"
1110         .by sp,sk,13
PRINT#15,"U2 2 0 11 7"
```

schreibt beispielsweise den aktuellen Pufferinhalt in den Sektor 7 auf der Spur 11. Damit haben wir alle Befehle zusammen, um uns einmal eine praktische Anwendung für den Direktzugriff anzusehen.

Wie Sie ja wissen, muß der Name einer Diskette beim Formatieren angegeben werden. Möchte man den Namen später ändern, muß die Diskette neu formatiert werden, wobei natürlich alle auf der Diskette gespeicherten Daten verloren gehen.

Wenn man nun aber weiß, daß der Diskettenname im ersten Directory-Block auf Spur 18, Sektor 0 gespeichert wird, kann man ihn mit Hilfe des Direktzugriffs leicht ändern, wie das folgende Programm zeigt:

```
100 rem namen einer diskette aendern
110 :
120 rem dateien oeffnen
130 open 15,8,15
140 open 2,8,2,"#"
150 :
160 rem block in puffer lesen
170 print#15,"u1 2 0 18 0"
180 rem auf diskettennamen positionieren
190 print#15,"b-p 2 144"
200 rem alten namen lesen und ausgeben
210 dt$=""
220 get#2,eg$
230 if eg$=chr$(160) then 260:rem ende des namens
240 dt$=dt$+eg$
250 goto 220
260 print "alter name: ";dt$:print
270 :
280 rem neuen namen erfragen
290 input "neuer name: ";nd$
300 nd$=left$(nd$,16):rem maximal 16 zeichen
310 rem namen mit 'shift'+'space' auffuellen
320 if len(nd$)<16 then nd$=nd$+chr$(160):goto 320
330 :
340 rem neuen namen in puffer schreiben
350 print#15,"b-p 2 144":zeiger setzen
360 print#2,nd$;:rem namen schreiben
370 rem block auf diskette zurueckschreiben
380 print#15,"u2 2 0 18 0"
390 :
400 rem dateien schliessen
410 close 2
420 close 15
```

Das Programm gibt zunächst den alten Namen der eingelegten Diskette aus, fragt dann nach dem gewünschten neuen Namen und schreibt diesen schließlich auf die Diskette. Wenn Sie jetzt

mit LOAD "$",8 das Directory der Diskette laden, werden Sie feststellen, daß der Name korrekt geändert wurde.

Einen Datenblock als frei/belegt kennzeichnen

Wie ich schon erwähnt habe, führt die Floppy für jede Diskette eine Blockbelegungstabelle, die BAM, in der alle belegten und freien Datenblöcke vermerkt sind. Wenn Sie im Direktzugriff auf einer Diskette Daten ablegen, müssen Sie sich darum nicht unbedingt kümmern. Es ist aber empfehlenswert, denn die BAM stellt auf jeden Fall die einfachste Möglichkeit dar, auf einer Diskette nicht den Überblick zu verlieren. Um einen Block in der BAM als "belegt" zu kennzeichnen, gibt es den "Block-Allocate-Befehl". Er hat die folgende Syntax:

```
PRINT#15,"B-A 0";SP;SK
```

bzw. in Assembler:

```
320           ldx #$0f        ;logische file-nr. 15
330           jsr $ffc9       ;datei auf ausgabe schalten
340           ldy #0
350 schleife  lda puffer,y    ;zeichen aus puffer holen
360           jsr $ffd2       ;und ausgeben
370           cmp #$0d        ;'return'?
375           beq fertig      ;ja, ausgabe beenden
380           iny             ;zeiger+1
385           jmp schleife    ;naechstes zeichen
390 fertig    nop             ;weiteres programm
395 ;
1100 puffer   .tx "b-a 0"
1110          .by sp,sk,13
```

SP und SK sind wiederum die Spur- und die Sektornummern des betreffenden Datenblocks. Mit Block-Allocate kann man auch leicht testen, ob ein Block, auf dem man Daten speichern möchte, überhaupt noch frei ist. Wenn Sie z.B. eingeben:

```
PRINT#15,"B-A 0 18 0"
```

dann erhalten Sie garantiert die Fehlermeldung NO BLOCK, da auf Spur 18, Sektor 0 das Directory beginnt. Um einen Block zu testen, versucht man ihn also als belegt zu kennzeichnen und liest dann den Fehlerkanal aus. Block-Allocate bietet dabei noch

einen besonderen Service: Falls der betreffende Block tatsächlich schon belegt ist, enthalten die letzten beiden Variablen der Fehlermeldung die Nummern des nächsten freien Datenblocks mit einer höheren Spur- und Sektornummer!

```
100 rem freien datenblock ermitteln
110 open 15,8,15
120 print#15,"b-a 0 0 0":rem block 0/0 belegen
130 input#2,fn,ft$,sp,sk:rem fehlermeldung
140 if fn=65 then print#15,"b-a 0";sp;sk
150 close 15
```

Mit dieser kleinen Routine können Sie bei Bedarf leicht einen freien Datenblock ermitteln. In Zeile 120 wird dazu versucht, den Block auf Spur 0, Sektor 0 als belegt zu kennzeichnen. Ist dieser Block noch nicht belegt, dann erhalten Sie die Meldung "00 OK 00 00". In diesem Fall enthalten SP und SK die korrekten Blocknummern 0 und 0.

Ist der Block dagegen schon belegt, enthalten SP und SK die Blocknummern des nächsten freien Blocks, den Sie dann zur Speicherung verwenden. In Zeile 140 wird der Block deshalb als belegt kennzeichnet. Um einen nicht mehr benötigten Datenblock wieder als frei kennzeichnen zu können, gibt es den "Block-Free-Befehl":

```
PRINT#15,"B-F 0";SP;SK
```

bzw. in Assembler:

```
320          ldx #$0f        ;logische file-nr. 15
330          jsr $ffc9       ;datei auf ausgabe schalten
340          ldy #0
350 schleife lda puffer,y    ;zeichen aus puffer holen
360          jsr $ffd2       ;und ausgeben
370          cmp #$0d        ;'return'?
375          beq fertig      ;ja, ausgabe beenden
380          iny             ;zeiger+1
385          jmp schleife    ;naechstes zeichen
390 fertig   nop             ;weiteres programm
395 ;
1100 puffer  .tx "b-f 0"
1110         .by sp,sk,13
```

Der Block-Free-Befehl macht also den Block-Allocate-Befehl wieder rückgängig. Bei Block-Free erhält man übrigens auch dann keine Fehlermeldung, wenn der betreffende Block bereits als frei gekennzeichnet war.

```
100 open 15,8,15
110 for z=0 to 20
120 print#15,"b-f 0";3;z
130 next z
140 close 15
```

beispielsweise gibt die gesamte Spur 3 frei. Das Belegen und Freigeben von Blöcken ist, wie gesagt, nur zur eigenen Kontrolle. Der Block-Write-Befehl kümmert sich um die Eintragungen in der BAM nicht. Wenn sie allerdings ein Programm oder eine sequentielle Datei auf der betreffenden Diskette speichern, werden die von Ihnen belegten Blöcke nicht überschrieben. Die Floppy-Routinen für die regulären Dateien sehen in der BAM nach, da die Blöcke der Dateien gekennzeichnet werden müssen.

Maschinenprogramme laden und starten

Eine Erweiterung des Block-Read-Befehls stellt der "Block-Execute-Befehl" dar. Block-Execute lädt einen Datenblock in den Pufferspeicher und führt ihn dann als Maschinenprogramm aus. Der Befehl ist wie folgt aufgebaut:

```
PRINT#15,"B-E 2 0";SP;SK
```

bzw. in Assembler:

```
320           ldx #$0f       ;logische file-nr. 15
330           jsr $ffc9      ;datei auf ausgabe schalten
340           ldy #0
350 schleife  lda puffer,y   ;zeichen aus puffer holen
360           jsr $ffd2      ;und ausgeben
370           cmp #$0d       ;'return'?
375           beq fertig     ;ja, ausgabe beenden
380           iny            ;zeiger+1
385           jmp schleife   ;naechstes zeichen
390 fertig    nop            ;weiteres programm
395 ;
1100 puffer   .tx "b-e 2 0"
1110          .by sp,sk,13
PRINT#15,"B-E 2 0 15 7"
```

beispielsweise lädt den Datenblock auf Spur 15, Sektor 7 und führt ihn aus. Mehr zum Thema Floppy-Speicher und Maschinenprogramme im nächsten Abschnitt.

5.7 Der Zugriff auf den Floppy-Speicher

Wie Sie ja bereits wissen, ist die Floppy mit einem eigenen Betriebssystem ausgestattet, das alle Diskettenoperationen weitestgehend selbständig, d.h. ohne die Unterstützung des Commodore 64 ausführt. Neben dem Betriebssystem, das im ROM gespeichert ist, existieren auch noch zwei KByte RAM, die normalerweise als Arbeits- und Pufferspeicher genutzt werden. Mit Hilfe der im folgenden vorgestellten Befehle ist es nun möglich, den Floppy-Speicher auszulesen (RAM+ROM) und in ihn hineinzuschreiben (RAM) sowie innerhalb des RAM-Speichers eigene Maschinenprogramme ausführen zu lassen. Alle die Speicherzugriffsbefehle werden über den Befehlskanal gesendet. Evtl. auszulesende Daten werden von der Floppy ebenfalls über den Befehlskanal bereitgestellt. Wir müssen also zu Beginn nur wie gewohnt den Befehlskanal mit

```
OPEN 15,8,15
```

bzw. in Assembler mit

```
140     lda #$0f        ;logische file-nr. 15
150     ldx #$08        ;geraeteadresse
160     ldy #$0f        ;sekundaeradr. 15
170     jsr $ffba       ;parameter setzen
175     jsr $ffc0       ;open-routine aufrufen
```

öffnen und am Ende nicht vergessen, ihn zu schließen. Mit

```
CLOSE 15
```

bzw.

```
920     jsr $ffcc       ;auf standardeingabe
                        ;zurückschalten
930     lda #$0f        ;logische file-nr.
940     jsr $ffc3       ;close-routine aufrufen
```

Daten aus dem Floppy-Speicher lesen

Um Daten aus dem Floppy-Speicher auszulesen, gibt es den "Memory-Read-Befehl".

Ihm wird die Speicheradresse, ab der gelesen werden soll, sowie die Anzahl der zu lesenden Bytes übergeben:

```
PRINT#15,"M-R" CHR$(LO) CHR$(HI) CHR$(AZ)
```

bzw. in Assembler:

```
320           ldx #$0f       ;logische file-nr. 15
330           jsr $ffc9      ;datei auf ausgabe schalten
340           ldy #0
350 schleife  lda puffer,y   ;zeichen aus puffer holen
360           jsr $ffd2      ;und ausgeben
370           cmp #$0d       ;'return'?
375           beq fertig     ;ja, ausgabe beenden
380           iny            ;zeiger+1
385           jmp schleife   ;naechstes zeichen
390 fertig    nop            ;weiteres programm
395 ;
1100 puffer   .by <(adresse),>(adresse),anzahl,13
```

LO und HI enthalten die Speicheradresse im bekannten Low-/High-Byte-Format. Sehen wir uns das ganze an zwei konkreten Beispielen an. Sehr nützlich ist es beispielsweise, die Anzahl der freien Blöcke der eingelegten Diskette zu ermitteln.

Dazu muß man wissen, daß die Floppy diesen Wert in den Speicherzellen 762 und 764 ablegt.

```
100 open 15,8,15
110 ad=762:hb=int(ad/256):lb=ad-hb*256:rem adresse zerlegen
120 print#15,"m-r" chr$(lb) chr$(hb) chr$(1)
130 get#15,bl$:bl=asc(bl$)
140 ad=764:hb=int(ad/256):lb=ad-hb*256:rem adresse zerlegen
150 print#15,"m-r" chr$(lb) chr$(hb) chr$(1)
160 get#15,bh$:bh=asc(bh$)
170 close 15
180 bf=bl+256*bh
```

Die Variable BF enthält nun die Anzahl der freien Blocks. Häufig möchte man von einem Programm aus kontrollieren, ob die

richtige Diskette im Laufwerk eingelegt ist. Am einfachsten geht das natürlich, indem man den Diskettennamen überprüft. Der Name der momentan eingelegten Diskette steht im Floppy-Speicher immer ab Adresse 1936, von wo wir ihn leicht auslesen können:

```
100 open 15,8,15
110 ad=1936:hb=int(ad/256):lb=ad-hb*256
    :rem adresse zerlegen
120 print#15,"m-r" chr$(lb) chr$(hb) chr$(16)
130 input#15,dn$
170 close 15
```

DN$ enthält jetzt den Diskettennamen, der allerdings mit dem ASCII-Code 160 (<Shift>+<Space>) auf 16 Zeichen aufgefüllt ist. Bevor man den Namen daher vergleicht, muß man die Vergleichsvariable ebenfalls auf 16 Zeichen auffüllen. Eine allgemeine Routine dazu könnte so aussehen:

```
200 vg$="disk1":rem diskettenname
210 if len(vg$)<16 then vg$=vg$+chr$(160):goto 210
```

Anschließend können Sie die beiden Namen mit

```
300 if dn$=vg$ then ...
```

miteinander vergleichen.

Daten in den Floppy-Speicher schreiben

Während man beim Auslesen des Floppy-Speichers kaum einen Schaden anrichten kann, muß man beim Schreiben (ähnlich wie bei den POKE-Befehlen in den Rechnerspeicher) sehr vorsichtig sein. Der Befehl zum Schreiben von Daten heißt "Memory-Write" und ist wie folgt aufgebaut:

```
PRINT#15,"M-W" CHR$(LO) CHR$(HI) CHR$(AZ) CHR$(D1) CHR$(D2) ...
```

bzw. in Assembler:

```
320        ldx #$0f        ;logische file-nr. 15
330        jsr $ffc9       ;datei auf ausgabe schalten
340        ldy #0
```

```
350 schleife  lda puffer,y   ;zeichen aus puffer holen
360           jsr $ffd2      ;und ausgeben
370           cmp #$0d       ;'return'?
375           beq fertig     ;ja, ausgabe beenden
380           iny            ;zeiger+1
385           jmp schleife   ;naechstes zeichen
390 fertig    nop            ;weiteres programm
395 ;
1100 puffer   .by <(adresse),>(adresse),anzahl
1110          .by daten-byte1,daten-byte2,...,13
```

LO und HI enthalten die Speicheradresse, ab der die Daten in der Floppy abgelegt werden sollen, im Low-/High-Byte-Format. Dahinter folgt die Anzahl der zu übertragenden Daten und schließlich die Daten selbst. Daß man mit Hilfe von Memory-Write in sehr elementare Abläufe der Floppy eingreifen kann, zeigt das folgende Beispiel:

Bevor die Floppy einen bestimmten Datenblock einer Diskette als fehlerhaft wertet, versucht sie normalerweise, fünfmal nacheinander auf ihn zuzugreifen. Erst wenn auch der fünfte Versuch scheitert, wird eine entsprechende Fehlermeldung gesendet. Die Anzahl der Zugriffsversuche ist nun in der Speicherzelle 106 gespeichert. Dieser Wert läßt sich mit Memory-Write natürlich ändern:

```
100 input "anzahl der versuche: "az
110 :
120 open 15,8,15
130 ad=106:hb=int(ad/256):lb=ad-hb*256:rem adresse zerlegen
140 print#15,"m-w" chr$(lb) chr$(hb) chr$(1) chr$(az)
170 close 15
```

Das Programm erlaubt es Ihnen, die Anzahl der Zugriffsversuche beliebig zu variieren.

Floppy-Routinen aufrufen

Ein Maschinenprogramm, das man mit Memory-Write in den Floppy-Speicher übertragen hat, läßt sich mit dem Befehl "Memory-Execute" starten. Das Programm muß dazu mit einem RTS abgeschlossen sein. Außerdem kann man mit Memory-Execute auch Unterprogramme des Floppy-Betriebssystems aufrufen. Der Befehl ist wie folgt aufgebaut:

```
PRINT#15,"M-E" CHR$(LO) CHR$(HI)
```

bzw. in Assembler:

```
320          ldx #$0f       ;logische file-nr. 15
330          jsr $ffc9      ;datei auf ausgabe schalten
340          ldy #0
350 schleife lda puffer,y   ;zeichen aus puffer holen
360          jsr $ffd2      ;und ausgeben
370          cmp #$0d       ;'return'?
375          beq fertig     ;ja, ausgabe beenden
380          iny            ;zeiger+1
385          jmp schleife   ;naechstes zeichen
390 fertig   nop            ;weiteres programm
395 ;
1100 puffer  .by <(adresse),>(adresse),13
```

LO und HI enthalten die Startadresse des Maschinenprogramms im Low-/High-Byte-Format. Das folgende Beispiel verzweigt ins Floppy-Betriebssystem und löscht den Kommandostring-Puffer der Floppy:

```
120 open 15,8,15
130 ad=49597:hb=int(ad/256):lb=ad-hb*256:rem adresse zerlegen
140 print#15,"m-e" chr$(lb) chr$(hb)
170 close 15
```

Damit haben wir alle Spezialbefehle durch. Wie sie gesehen haben, ist die Handhabung dieser Befehle zwar nicht ganz leicht, dafür erlauben sie aber zum Teil sehr interessante Effekte.

5.8 Diskettenkopierschutz

Wie arbeitet ein solcher Diskettenkopierschutz? Wie und wofür kann ich ihn selber benutzen? Das sind die Fragen, die auf den folgenden Seiten beantwortet werden sollen.

Wofür Sie einen Kopierschutz brauchen, ist einfach zu beantworten. Sie können mit Hilfe eines solchen Schutzes sicherstellen, daß Ihr mühsam geschriebenes Programm nicht ohne Ihr Wissen weitergegeben werden kann. Diese in diesem Buch erläuterten Schutzverfahren sind auch ohne weiteres dazu geeignet, professionelle Software zu schützen. Sie entsprechen nicht den in

einigen Zeitschriften oder ähnlichem beschriebenen Verfahren, die mit jedem drittklassigen Kopierprogramm kopiert werden können. Selbst wenn Sie kein Interesse haben, Ihre Software zu schützen, ist dies auch im Hinblick auf die interessante Floppy-Programmierung sehr zu empfehlen.

Auf die Frage, wie ein solcher Kopierschutz arbeitet, ist allgemein zu sagen, daß es sich hierbei um eine Manipulation auf der geschützten Diskette handelt, die nicht ohne weiteres von einem Kopierprogramm kopiert werden kann. Wie dieses im einzelnen aussieht, wird im Laufe dieses Buches noch besprochen.

Leider müssen wir zum Verständnis der folgenden Programme ein recht gutes Wissen über das Arbeiten der Floppy sowie Kenntnisse der Programmierung in Maschinensprache voraussetzen. Für diejenigen, die diese Kenntnisse nicht haben, ist dies dennoch interessant, weil alle Schutzsysteme auch in Form eines BASIC-Loaders vorliegen und somit für jeden nutzbar sind.

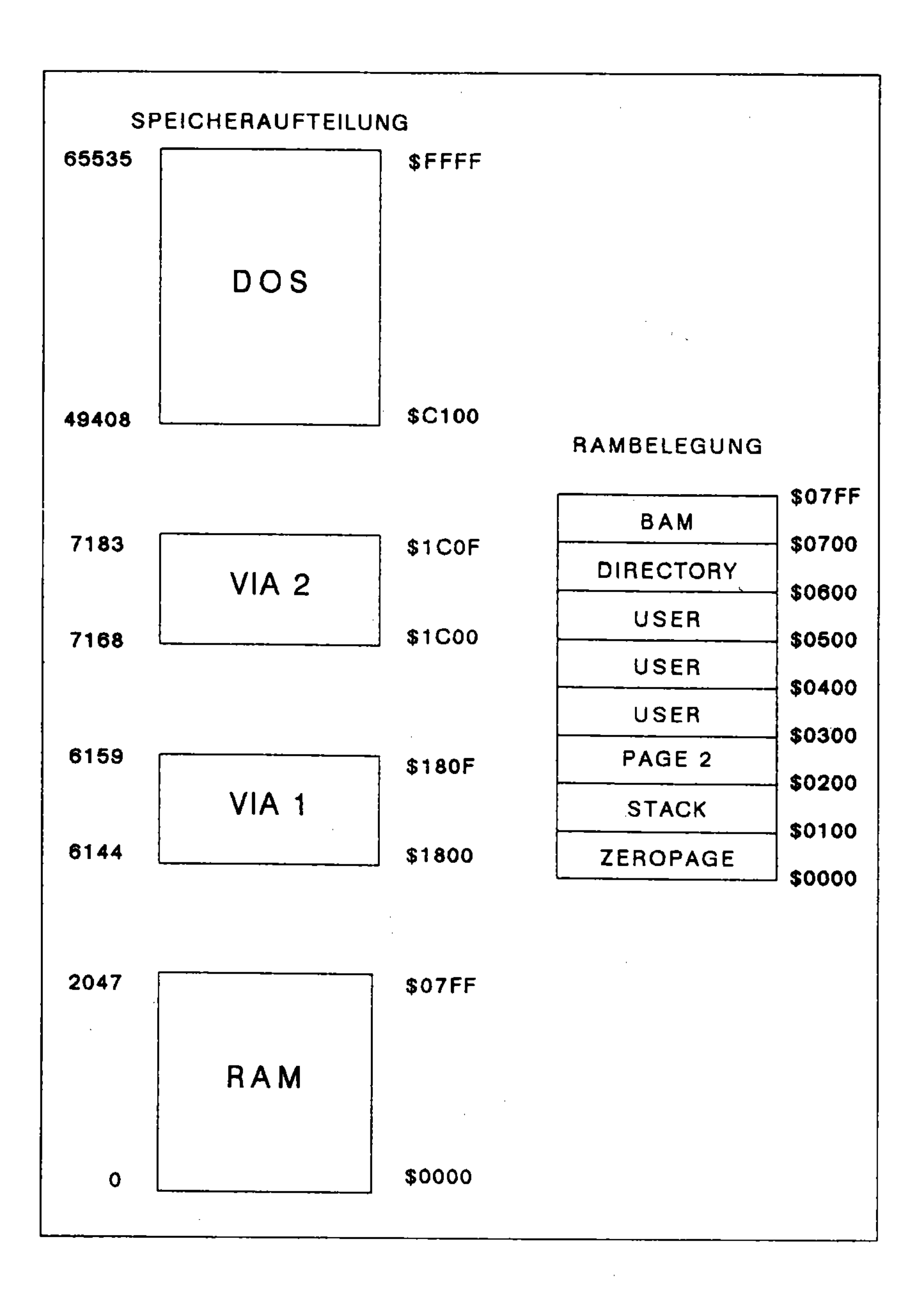
Unser "Handwerkszeug"

Bevor wir uns intensiver mit der Programmierung eines Kopierschutzes befassen, wollen wir zuvor näher auf das "Innenleben" der Floppy eingehen, um Ihnen das Verständnis der zu besprechenden Kopierschutzsysteme zu ermöglichen. Wenn Sie über dieses Wissen bereits verfügen, können Sie sich sofort dem Auftragen eines Kopierschutzes widmen. An dieser Stelle nun die Übersicht über die Speicheraufteilung der Floppy und eine Übersicht über die Nutzung der Puffer der Floppy.

VIA 1 ist der Baustein, der den Datenaustausch zwischen Computer und Rechner herstellt. Er verwaltet den seriellen Bus.

VIA 2 wird ausschließlich für den Datenaustausch zwischen der Floppy und der Diskette verwendet.

Es ist noch wissenswert, daß das RAM der Floppy ab $8000 gespiegelt ist, was bedeutet, daß die Speicherstellen ab $8000 die gleichen Werte und auch die gleiche Bedeutung wie die Speicherstellen ab $0000 haben.


SPEICHERAUFTEILUNG
65535
$FFFF
DOS
49408
$C100
7183
$1C0F
VIA 2
7168
$1C00
6159
$180F
VIA 1
6144
$1800
2047
$07FF
RAM
0
$0000
RAMBELEGUNG
$07FF
BAM
$0700
DIRECTORY
$0600
USER
$0500
USER
$0400
USER
$0300
PAGE 2
$0200
STACK
$0100
ZEROPAGE
$0000

Die Arbeitsweise des DOS

Die VC1541 ist ein "intelligentes" Diskettenlaufwerk mit eigenem Mikroprozessor und Betriebssystem (Disk Operating System, DOS). Dadurch wird kein Speicherplatz und keine Rechenzeit des angeschlossenen Rechners benötigt. Der Rechner braucht der Floppy lediglich Befehle zu übermitteln, die diese dann selbsttätig ausführt.

Die Floppy hat damit drei Aufgaben gleichzeitig zu erledigen: Zum einen muß sie den Datenverkehr vom und zum Rechner durchführen. Die zweite Aufgabe ist die Interpretation der Befehle, die Verwaltung von Dateien, der zugeordneten Übertragungskanäle und der Blockpuffer. Die dritte Aufgabe ist die hardware-mäßige Bedienung der Diskette; dazu gehört das Schreiben und Lesen einzelner Blocks auf der Diskette sowie das Formatieren von Disketten. Diese Aufgaben muß bei der VC1541 ein 6502-Mikroprozessor gleichzeitig durchführen. Dies ist nur mit Hilfe der Interrupt-Technik möglich. Nur so können drei Programme quasi gleichzeitig ablaufen.

Nachdem wir schon etwas näher auf den Aufbau der Diskette eingegangen sind, wollen wir uns einmal ansehen, wie das DOS alle seine Aufgaben realisiert. Dies ist besonders dann wichtig, wenn man selbst Programme in der Floppy schreiben will, wie beispielsweise seinen eigenen Kopierschutz.

Beim Schreiben von Programmen im Floppy-puffer kann man selbstverständlich auch viel vom Betriebssystem der Floppy Gebrauch machen, das für die meisten Aufgaben schon Routinen parat hat. Doch in der Floppy ist dies nicht so problemlos wie im C64. Das Betriebssystem der Floppy kann man in zwei Teile unterteilen. Der erste Teil ist das Hauptprogramm, das in einer Endlosschleife läuft. Von diesem Programm aus wird hauptsächlich der serielle Bus verwaltet. Fast alle Unterprogrammaufrufe werden mit absoluten Sprüngen realisiert und müssen somit auch mit einem JMP-Befehl zurück ins Hauptprogramm abgeschlossen werden. Diese Routinen können deswegen kaum für die eigene Arbeit verwendet werden, da sie nach einmaligem Aufruf sofort ins Hauptprogramm springen. Sie müssen solche Routinen also

selbst schreiben und können nicht auf die vorhandenen Routinen des Betriebssystems zurückgreifen.

Der zweite Teil des Betriebssystems ist dafür um so besser für eigene Programme zu verwenden. Es handelt sich hierbei um ein Interrupt-Programm, die sogenannte "Job-Schleife".

Dieses Programm übermimmt die Lese- und Schreiboperationen auf und von Diskette. Bei jedem Interrupt werden die Speicherstellen $00 bis $05 der Zeropage, die die "Schnittstelle" zwischen dem Hauptprogramm und Interrupt-Programm herstellen, auf ihre Werte überprüft. Werte, die größer oder gleich $80 (128) sind, werden als Befehle (Jobs) erkannt und daraufhin ausgeführt. Jede dieser Speicherstellen (Job-Speicher) bezieht sich auf einen bestimmten Speicherbereich. Zusätzlich gehören zu jedem Job-Speicher noch zwei Speicherstellen, die den Track und den Sektor angeben, auf die sich der Job (Befehl) bezieht. Nachfolgend eine Tabelle, welche den Job-Speichern ihre Track- und Sektorangabe sowie ihren Speicherbereich zuordnet:

Job	Track	Sektor	Speicherbereich
$00	$06	$07	$0300-$03FF
$01	$08	$09	$0400-$04FF
$02	$0A	$0B	$0500-$05FF
$03	$0C	$0D	$0600-$06FF
$04	$0E	$0F	$0700-$07FF
$05	$10	$11	kein RAM

Anschließend folgt eine Tabelle, die die Job-Codes und ihre Bedeutung zeigt.

Job-Code	Bedeutung
$80 128	Sektor lesen
$90 144	Sektor schreiben
$A0 160	Sektor verifizieren
$B0 176	Sektor suchen
$C0 192	Bump, Kopfanschlagen
$D0 208	Programm im Puffer ausführen
$E0 224	Programm im Puffer ausführen, vorher Laufwerksmotor einschalten und Kopf positionieren

Machen wir uns nun die Verarbeitung der Job-Codes anhand eines einfachen Beispiels besser verständlich:

Wenn vom BASIC aus durch einen U1-Befehl ein Block gelesen werden soll, so nimmt das Hauptprogramm diesen Befehl entgegen, erkennt ihn und veranlaßt, daß der entsprechende Block gelesen wird. Diese Veranlassung sieht jetzt so aus, daß die Track- und Sektornummer in die Speicherstellen, die im Zusammenhang mit dem angegebenen Puffer stehen, geschrieben werden. Daraufhin wird der Job-Code $80 (Sektor lesen) in den entsprechenden Job-Speicher geschrieben.

Nehmen wir an, zum Lesen dieses Blocks wäre Puffer 2 reserviert worden; dann wären die Track- und Sektornummer in die Speicherstellen $0A und $0B geschrieben worden. Der Job-Code $80 wäre danach in den Job-Speicher $02 geschrieben worden. Nach diesen Arbeitsgängen beginnt die Aufgabe der Job-Schleife. Diese überprüft jetzt die Job-Speicher, findet im Job-Speicher die $80 und holt sich aus $0A und $0B die Track- und Sektornummer des zu lesenden Sektors. Daraufhin wird der Kopf positioniert und der Sektor gelesen. Die gelesenen Daten werden im Puffer 2 ($0500-$05FF) abgelegt.

Das Hauptprogramm befindet sich während dieser Zeit in einer Wartestellung. Es wartet auf die READY-Meldung des Jobs. Diese Rückmeldung wird in denselben Job-Speicher geschrieben, in den auch der Befehl geschrieben wurde. Die Rückmeldungen unterscheiden sich von den Job-Codes dadurch, daß bei den Job-Codes das höchstwertige Bit gesetzt und bei den Rückmeldungen gelöscht ist. Das Hauptprogramm überprüft immer das höchstwertige Bit und beginnt, sobald die Rückmeldung kommt, diese auszuwerten.

Bevor wir auf die Rückmeldungen eingehen, noch ein Hinweis zu den Job-Codes, die ein Programm im Puffer ausführen: Der Job-Code $D0 startet ein Programm bei der Anfangsadresse des jeweiligen Puffers. Der Befehl $E0 fährt den Laufwerksmotor zuvor noch hoch, positioniert den Kopf und führt dann wie $D0 das Programm im angegebenen Puffer aus.

Wollen wir also den Inhalt von Track 24, Sektor 8 in den Puffer #0 ($300 - $3FF) lesen, kann dies durch folgende Routine geschehen:

```
100 OPEN 1,8,15
110 TRACK = 24: SECTOR = 8: LESEN = 128
120 PRINT#1, "M-W"CHR$(6)CHR$(0)CHR$(2)CHR$(TRACK)CHR$(SECTOR)
130 PRINT#1, "M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(LESEN)
140 PRINT#1, "M-R"CHR$(0)CHR$(0)
150 GET#1, A$:IF A$="" THEN A$=CHR$(0)
160 IF ASC(A$)>127 THEN 140
170 PRINT "FEHLERCODE ="; ASC(A$)
180 CLOSE 1
```

In Zeile 120 werden Track- und Sektornummer für Puffer #0 übergeben, in Zeile 130 der Befehl zum Lesen. Die Programmschleife in Zeile 140 bis 160 wartet das Ende des Jobs ab. In Zeile 170 wird der Fehlercode ausgegeben. Diese Codes haben folgende Bedeutung:

Code	Bedeutung	DOS-Fehlermeldung
$01	Alles OK	00, OK
$02	Headerblock nicht gefunden	20, READ ERROR
$03	SYNC nicht gefunden	21, READ ERROR
$04	Datenblock nicht gefunden	22, READ ERROR
$05	Checksummenfehler im Datenblock	23, READ ERROR
$07	Verify-Fehler	25, WRITE ERROR
$08	Diskette schreibgeschützt	26, WRITE PROTECT ON
$09	Checksummenfehler im Headerblock	27, READ ERROR
$0B	Falsche ID gelesen	29, ID MISMATCH
$0F	Diskette nicht eingelegt	74, DRIVE NOT READY

Die im Floppyhandbuch aufgeführten Fehlermeldungen 24, READ ERROR (GCR-Kodierung wird nicht erkannt) und 28, WRITE ERROR treten bei der VC1541 nicht auf. Zur Bedeutung der Fehlermeldungen sehen Sie bitte im Anhang nach.

Zur Benutzung der Disk-Controller-Routinen sei noch folgendes vermerkt: Wenn Sie einen Block mit dem U1-Befehl lesen, so wird der Befehl "Lese Block" an die Disk-Controller übergeben. Wird von diesem nun ein Fehler gemeldet, so veranlassen die "logischen DOS-Routinen" (das Hauptprogramm) weitere Leseversuche jeweils eine halbe Spur links und rechts vom Track.

Hat auch dies keinen Erfolg, wird ein "Bump" durchgeführt (der Kopf geht nach Track 1 und gibt die typischen Geräusche bei einem Lesefehler von sich) und das ganze wird nochmal versucht. Normalerweise werden vom DOS fünf Leseversuche gemacht, ehe endgültig ein Fehler gemeldet wird. Diesen Mechanismus umgehen wir, wenn wir direkt die Disk-Controller-Routinen ansprechen. Für normale Schreib- und Leseoperationen sollten daher weiterhin die U1- und U2-Befehle benutzt werden.

5.8.1 Das Disketten-Aufzeichnungsverfahren

Hier versuchen wir, Ihnen näher zu bringen, wie die Bits auf die Diskette kommen und von dort wieder gelesen werden können. Wie Sie bereits wissen, ist die Diskette in 35 Spuren oder Tracks unterteilt, die als konzentrische Ringe auf der Diskette angeordnet sind. Die äußerste Spur erhält die Nummer 1, und Track 35 ist die innerste Spur. Um die einzelnen Spuren anzusteuern, hat das Laufwerk einen Schritt- oder Steppermotor, mit dem der Schreib-/Lesekopf über jeden Track positioniert werden kann. Die Aufzeichnung der Daten geschieht nun bitweise. Dabei wird jedes 1-Bit durch einen Wechsel der Magnetisierungsrichtung gekennzeichnet, während bei einem 0-Bit nichts passiert. Die Bits werden in einem festen Takt geschrieben, der ca. 250000 Bits/s beträgt.

Wenn wir nun Daten nach diesem Verfahren auf eine Spur schreiben, sind wir anschließend nicht mehr in der Lage, die Daten korrekt wieder zu lesen. Da sich die Diskette fortlaufend dreht, die Spuren also keinen Anfang und kein Ende haben, können wir den Beginn unserer Aufzeichnung nicht ermitteln. Doch selbst wenn wir den Anfang der Spur feststellen könnten, würden wir Probleme haben, den Beginn jedes Bytes festzustellen. Theoretisch ist durch den Beginn der Aufzeichnung auch der Beginn jedes einzelnen nachfolgenden Bytes bestimmt. Dies setzt jedoch voraus, daß die Umdrehungsgeschwindigkeit der Diskette immer absolut gleichbleibend ist, auch von einem Drive zum anderen. Diese Forderung ist jedoch illusorisch, und man hat sich eine andere Möglichkeit überlegt, wie man den Beginn der einzelnen Bytes sicher erkennen kann.

Zuerst unterteilt man einen Track in mehrere Sektoren. Da auf einer Spur bei der oben genannten Bit-Rate mehr als 6000 Bytes untergebracht werden können, braucht man zum einen einen Pufferspeicher dieser Größe in der Floppy. Ein weiterer Nachteil wäre es, daß nur maximal 35 Dateien abzuspeichern wären. Wenn wir ein Programm von 1 KByte Länge abspeichern würden, so wären 5 KByte verschenkt, die Ausnutzung der Diskette also sehr unökonomisch. Da die Tracks von außen nach innen schmaler werden, werden die äußeren Tracks in mehr Sektoren unterteilt als die inneren. Dazu muß die Bit-Rate nach außen hin gesteigert werden. Die folgende Tabelle enthält die entsprechenden Daten.

Track	Bit-Rate	Bytes pro Track
1-17	307692	7692
18-24	285714	7143
25-30	266667	6667
31-35	250000	6250

Doch damit hat sich unser Problem nur vergrößert: Wie erkennen wir den Beginn der Sektoren auf dem Track? Wir müssen also bestimmte Bit-Kombinationen bevorzugt erkennen können, die als Daten-Bytes nicht vorkommen können. Mit acht Bit lassen sich 256 Kombinationen darstellen, die es auch alle als Daten-Bytes kann. Der Schlüssel zur Lösung liegt darin, ein Byte nicht durch 8, sondern für die Diskettenaufzeichnung durch mehr Bits darzustellen. Damit sind wir schon beim "Group Code Recording", wie es von Commodore verwendet wird.

Wir benutzen als sogenanntes Synchron- oder SYNC-Zeichen die Bit-Kombination %11111111 oder $FF. Die Daten-Bytes müssen nun so verschlüsselt werden, daß niemals acht 1-Bits hintereinander vorkommen. Das "Group Code Recording", kurz GCR genannt, bildet nun jeweils 4 Bits auf 5 Bits ab. Die 5-Bit-Kombination ist so gewählt, daß die obige Forderung erfüllt ist. Eine zusätzliche Forderung besteht noch darin, dafür zu sorgen, daß auch niemals mehr als zwei 0-Bit hintereinander folgen, da dabei ja kein Wechsel der Magnetisierungsrichtung erfolgt und es

dadurch Probleme beim Lesen der Daten geben kann. Die Tabelle gibt nun an, wie jeweils 4 Bits verschlüsselt werden.

Originaldaten		GCR-Code
$0	0000	01010
$1	0001	01011
$2	0010	10010
$3	0011	10011
$4	0100	01110
$5	0101	01111
$6	0110	10110
$7	0111	10111
$8	1000	01001
$9	1001	11001
$A	1010	11010
$B	1011	11011
$C	1100	01101
$D	1101	11101
$E	1110	11110
$F	1111	10101

Durch beliebiges Aneinandereihen des GCR-Codes kann also niemals ein Bit-Muster entstehen, das mehr als 8 aufeinanderfolgende 1-Bits oder mehr als zwei 0-Bits enthält. Wir können also jetzt den Beginn eines Sektors durch SYNC-Bytes markieren, die durch eine Digitalschaltung erkannt und gemeldet werden. Doch wie wissen wir, welcher Sektor nach einem SYNC-Byte folgt? Dazu steht vor jedem Sektor ein sogenanntes BlockHeader-Feld (Sektorkopf), das Track- und Sektornummer sowie zusätzlich noch die Identifikation der Diskette enthält (das sind die zwei Zeichen, die Sie beim Formatieren der Diskette mit angeben). Der nachfolgende eigentliche Sektorinhalt wird wieder durch vorausgehende SYNC-Bytes eingeleitet. Damit man Blockheader und Datenblock unterscheiden kann, folgt nach dem SYNC-Zeichen ein Kenn-Byte. Ein Headerblock wird durch eine 8 identifiziert, ein Datenblock durch eine 7. Damit Lesefehler, die z.B. durch eine verschmutzte oder zerkratzte Diskette entstehen können, erkannt werden, enthält jeder Block noch eine Checksumme über alle Bytes. Ein Blockheader hat damit folgenden Aufbau:

```
5 Sync-Zeichen
1 Headerblock-Kennzeichen ($08)
1 Checksumme über alle Bytes
1 Sektornummer
1 Track-Nummer
1 ID2
1 ID1
2 Abschluß-Bytes ($0F)
8 Füll-Bytes ($55)
```

Bis auf die SYNC-Zeichen ($FF, %11111111) und die 8 Füll-Bytes ($55 %01010101) werden alle anderen Bytes in GCR-Code umgewandelt. Die Füll-Bytes sind erforderlich, da immer nur eine durch 4 teilbare Anzahl Bytes in GCR-Code umgewandelt werden kann. Aus einem Byte (8 Bits) werden nach der Umwandlung 10 Bits. 4 Bytes (32 Bits) werden in 40 Bits umgewandelt, was gerade 5 Bytes entspricht. Das folgende Schema soll deutlich machen, wie die Umwandlung vonstatten geht.

```
Originaldaten   11112222 33334444 55556666 77778888
GCR-Daten       11111222 22333334 44445555 56666677 7778888
```

Eine Zifferngruppe soll dabei ein "Nibble" (4 Bits) darstellen. Die Umwandlung geschieht von einer DOS-Routine mit Hilfe der GCR-Code-Tabelle, die ab Adresse $F77F im ROM steht.

Doch jetzt zum Aufbau des Datenblocks.

```
     5  Sync-Zeichen
     1  Datenblock-Kennzeichen ($07)
   256  Daten-Bytes
     1  Checksumme
     2  Abschluß-Bytes ($00)
MIN. 4  Füll-Bytes ($55)
```

Der Block wird wieder mit 5 SYNC-Zeichen eingeleitet, dann folgen im GCR-Code das Kennzeichen für den Datenblock, eine 7, dann die 256 Daten-Bytes. Zur Fehlererkennung folgt eine Ein-Byte-Checksumme (EXKLUSIV-ODER-VERKNÜPFUNG aller Bytes) und zwei Null-Bytes als Abschluß-Bytes. Die Anzahl der Füll-Bytes bis zum Beginn des nächsten Headerblocks ist variabel und wird bei der Formatierung aus der Gesamtkapazität des jeweiligen Tracks ermittelt.

Ein kompletter Sektor auf Diskette enhält also 8 (Headerblock) + 260 (Datenblock) Bytes, die in GCR-Code gewandelt 335 Bytes ergeben, sowie 10 SYNC-Zeichen und mindestens 12 Füll-Bytes. Damit besteht ein Sektor aus mindestens 357 Bytes im Verhältnis zu 256 Nutz-Bytes. Es gehen also ca. 28% der gesamten Diskettenkapazität für die Datenorganisation verloren.

Das Beschreiben der Diskette geschieht bereits komplett beim Formatieren. Sehen wir uns jetzt einmal an, wie die Floppy einen Sektor liest.

1. Laufwerksmotor einschalten.
2. Schreib-/Lesekopf über Track bringen.
3. Headerblock für gewünschten Track und Sektor erzeugen und in GCR-Code umwandeln.
4. SYNC-Zeichen abwarten.
5. Die nächsten 8 Bytes lesen und mit dem oben erzeugten GCR-Code vergleichen.
6. Falls keine Übereinstimmung: zurück zu Schritt 4.
7. Der richtige Sektorheader wurde gefunden, jetzt SYNC-Zeichen des Datenblocks abwarten.
8. Datenblock lesen und GCR-Code decodieren.

Dies sind, vereinfacht dargestellt, die Schritte, die beim Lesen eines Sektors durchlaufen werden. Nicht berücksichtigt wurde dabei, was im Fehlerfall passiert, z.B. falls ein Checksummenfehler festgestellt wurde oder falls schon der Headerblock nicht korrekt gelesen werden konnte.

Das Schreiben eines Blocks verläuft ähnlich. Auch hier muß zuerst der korrekte Headerblock gefunden werden, ehe man den alten Datenblock mit den neuen Daten überschreiben kann.

1. Laufwerksmotor einschalten.
2. Schreib-/Lesekopf über Track bringen.
3. Datenblock in GCR-Code umwandeln.
4. Headerblock für gewünschten Track und Sektor erzeugen und in GCR-Code umwandeln.
5. SYNC-Zeichen abwarten.

6. Die nächsten 8 Bytes lesen und mit dem oben erzeugten GCR-Code vergleichen.
7. Falls keine Übereinstimmung: zurück zu Schritt 5.
8. 8 GAP-Bytes überlesen.
9. 5 SYNC-Zeichen schreiben.
10. GCR-Daten schreiben.
11. Geschriebenen Sektor lesen und mit den Daten im RAM vergleichen (Verify).

Beim Schreiben eines Blocks wird also nur der Datenblock mit den zugehörigen SYNC-Bytes geschrieben. Der Blockheader bleibt unverändert, er wird nur einmal bei der Formatierung erzeugt.

5.8.2 Einführung in die Lese- und Schreibtechnik

Bevor wir dazu übergehen, die einzelnen Kopierschutzverfahren zu erläutern, sollte erst grundlegend die Methodik des Lesens und Speicherns von Daten auf Diskette besprochen werden.

Die zur Floppy-Steuerung interessantesten Register sind die Register 1 und 2 des VIA 2. Es sind die Adressen $1C00 und $1C01, wobei $1C00 hauptsächlich zur Ansteuerung der Laufwerk-LED und der Motoren und $1C01 zum Lesen und Schreiben der Daten benötigt werden.

Erläuterung der Bit-Funktionen der Adresse $1C00:

Bit	Funktion	
0	Steppermotorsteuerung	
1	Steppermotorsteuerung	
2	Laufwerksmotor	Bit=0: Motor aus
3	Laufwerk-LED	Bit=0: LED aus
4	Schreibschutz	Bit=0: Lichtschranke unterbrochen
5	Einstellung der Bit-Rate zum Tonkopf, Speed-Flag	
6	Bedeutung wie Bit 5	
7	SYNC-Signal beim Lesen gefunden	Bit=0: SYNC gefunden

Zum Lesen und Schreiben von Daten wird, wie schon gesagt, $1C01 benutzt. Schreibt man einen Wert in dieses Register, so wird dieser, sofern zuvor auf "Schreiben" umgestellt wurde, dieses Byte auf die Diskette geschrieben. Doch wie kann der Programmierer feststellen, wann das Byte vollständig geschrieben oder gelesen wurde? Um dies feststellen zu können, existiert eine Byte-Ready-Leitung, die mit dem Overflow-Flag des Prozessor-Status-Registers verbunden ist. Sobald ein Byte vollständig übertragen wurde, wird aufgrund dieser Leitung das Overflow-Flag gesetzt. Dieses Flag wird von dieser Leitung zwar gesetzt, jedoch nicht wieder gelöscht. Dies muß der Programmierer mit Hilfe des CLV-Befehls "von Hand" erledigen, damit er nicht ständig die Ready-Meldung für ein verarbeitetes Byte erhält. Die Zeit, in der ein Byte von Diskette gelesen oder auf Diskette geschrieben wird, liegt zwischen 26 und 32 Mikrosekunden, je nach Einstellung der Speed-Flags.

Zum Starten von Programmen in der Floppy gibt es grundsätzlich zwei Möglichkeiten. Zum einen können Sie Ihre Programme über die Job-Codes $E0 und $D0 als Interrupt-Programm starten, zum anderen ist es auch möglich, die Programme direkt zu starten. Wir werden uns erst das Starten von Programmen mit Hilfe der Job-Codes ansehen. Beispiel für das Laden eines Bytes:

```
0500 BVC $0500
0502 CLV
0503 LDA $1C01
```

In der ersten Zeile wird so lange gewartet, bis das V-Flag (Byte-Ready) auf High gesetzt wird als Zeichen dafür, daß ein Byte anliegt. Das V-Flag wird daraufhin "von Hand" wieder auf Low gesetzt. Anschließend wird das geladene Byte in den Akku geholt. Beispiel für das Speichern eines Bytes:

```
0500 STA $1C01
0503 BVC $0503
0505 CLV
```

Das Speichern eines Bytes läuft, wie Sie sehen, analog dazu ab. Das Byte wird gespeichert, worauf auf Byte-Ready gewartet wird. Abschließend wird das V-Flag wieder gelöscht. Vor dem

Speichern muß man der Floppy noch mitteilen, daß die Daten, die in die Adresse $1C01 geschrieben werden, auch wirklich gespeichert werden. Dies erledigt das folgende kleine Programm.

```
LDA $1C0C
AND #$1F     PCR auf Schreibbetrieb umschalten
ORA #$C0
STA $1C0C
LDA #$FF     Port für Schreib-/Lesekopf
STA $1C03    auf Ausgang
```

Natürlich gibt es auch eine Routine zum Umschalten auf Lesen:

```
LDA $1C0C
ORA #$E0     PCR auf Lesebetrieb umschalten
STA $1C0C
LDA #$00     Port für Schreib-/Lesekopf
STA $1C03    auf Eingang
```

Diese Routine existiert auch in dieser Form im Betriebssystem der Floppy und läßt sich aufrufen mit:

```
JSR $FE00
```

Um Daten gezielt auf Diskette schreiben und auch wieder von dort lesen zu können, wird es häufig nötig sein, auf eine SYNC-Markierung zu warten. Auch für diesen Fall stellt das Betriebssystem der Floppy eine Routine zur Verfügung. Sie läßt sich aufrufen mit:

```
JSR $F556
```

Die Routine wartet 20 ms auf ein SYNC-Signal. Wird dieses nicht gefunden, wird eine entsprechende Fehlermeldung ausgegeben. Hier liegt auch der Nachteil im Benutzen der Routine. Der Programmierer hat keine Möglichkeit, selbst auf ein eventuelles Nichtauffinden der SYNC-Markierung zu reagieren, da kein Rücksprung auf den Unterprogramm-Aufruf folgt.

Die Lösung ist einfach. Sie übernehmen die Routine in Ihr Programm und können somit bei negativer Abfrage zu Ihrer eigenen Routine verzweigen. Wenn Sie sicher sind, daß eine SYNC-Mar-

kierung gefunden wird, können Sie das entsprechende Bit auch ohne Fehlerabfrage direkt abfragen.

```
0500 BIT $1C00  Port abfragen
0502 BMI $0500  verzweige, wenn SYNC nicht gefunden
```

Das folgende Programm arbeitet selbst mit Job-Codes und wird daher mit einem M-E-Befehl aufgerufen. Es erzeugt einen 22 READ-ERROR auf dem eingestellten Track und Sektor.

```
0600 LDA #$01   Nummer des zu ladenden Tracks
0602 STA $0A    in Track-Speicher für Puffer 2 schreiben
0604 LDA #$00   Nummer des zu ladenden Sektors
0606 STA $0B    in Sektorspeicher für Puffer 2 schreiben
0608 LDA #$80   Job-Code für Track/Sektor lesen
060A STA $02    in Job-Speicher des Puffer 2 schreiben
060C LDA $02    Job-Speicher lesen
060E BMI $060C  warten, bis Job abgearbeitet ist
0610 LDA #$09   eigenes Erkennungszeichen für Datenblock
0612 STA $47    in die entsprechende Adresse schreiben
0614 LDA #$90   Job-Code für Block speichern
0616 STA $02    in Job-Speicher für Puffer 2 schreiben
0618 LDA $02    Job-Speicher lesen
061A BMI $0618  warten, bis Job abgearbeitet ist
061C LDA #$07   normales Datenblock-Erkennungszeichen laden
061E STA $47    und speichern (rücksetzen)
0620 RTS        Rücksprung
```

In unserem Fall wird die Erkennungsziffer 7 durch die Ziffer 9 ersetzt. Der Daten-Header kann somit nicht mehr gefunden werden. Bei diesem ERROR handelt es sich um einen SOFT-ERROR, da die Daten des zerstörten Blocks sich trotz einer Fehlermeldung im Speicher der Floppy befinden. Vom BASIC oder von einem gewöhnlichen Diskmonitor aus ist es nicht möglich, diesen Block ohne Umstellung der Speicherzelle $47 (71) auf 9 zu lesen.

Falls Sie diesen Block wieder reparieren wollen, reicht es aus, 0610 LDA #$09 durch 0610 LDA #$07 zu ersetzen und das Programm erneut zu starten. Sie speichern somit die Orginalkennzahl 7 mitsamt dem Block wieder ab.

Als nächstes stellen wir eine Routine vor, die den Tonkopf um beliebig viele Halbspuren nach innen oder außen bewegt. In

diesem Beispiel wird der Tonkopf 16 Spuren nach außen und danach wieder nach innen gefahren. Es ist ratsam, die Diskette vor dem Ausprobieren dieser Routine zu initialisieren, damit der Tonkopf nicht anstoßen kann. Für die Beobachtung des Tonkopfes bei diesem Beispiel und bei eigenen Experimenten ist es von Vorteil, die vier Schrauben an der Unterseite der Floppy zu lösen und den Deckel vorsichtig abzunehmen. Unserer Meinung nach überwiegen die Vorteile beim Arbeiten mit einer geöffneten Floppy im Gegensatz zu einer geschlossenen. Jetzt aber endlich das erwähnte Programm.

```
0500 SEI         Interrupt sperren
0501 LDA $1C00   Control-Register laden
0504 ORA #$04    Bit für Laufwerksmotor setzen
0506 STA $1C00   und wieder schreiben
0509 LDX #$30    Warteschleife
050B STX $4B     um dem
050D DEC $4B     Motor Zeit
050F BNE $050D   zum Hochfahren
0511 DEX         zu geben
0512 BNE $150D   verzweige, wenn Zeit nicht abgelaufen
0514 LDY #$10    Zähler für Stepanzahl auf $10 setzen
0516 JSR $0530   Kopf um eine Halbspur nach außen schieben
0519 DEY         Zähler verringern
051A BNE $0516   verzweige, wenn noch nicht abgezählt
051C LDY #$10    Zähler erneut setzen
051E JSR $0537   Kopf um eine Halbspur nach innen schieben
0521 DEY         Zähler verringern
0522 BNE $051E   verzweige, wenn noch nicht abgezählt
0524 LDA $1C00   Control-Register laden
0527 AND #$F8    Bits 0 bis 2 löschen, um Motor zu stoppen
0529 STA $1C00   und wieder schreiben
052C CLI         Interrupt wieder ermöglichen
052D RTS         Rücksprung
052E NOP
052F NOP
```

Routine zur Kopfsteuerung

```
0530 LDX $1C00   Control-Register laden
0533 DEX         verringern, um Kopf nach außen zu schieben
0534 CLC         Carry-Flag setzen
0535 BCC $053B   unbedingter Sprung

0537 LDA $1C00   Control-Register laden
053A INX         erhöhen für Kopfbewegung nach außen
053B TXA         Wert in Akku schieben
053C AND #$03    ersten zwei Bits isolieren
053E STA $4B     und zwischenspeichern
```

```
0540 LDA $1C00  Control-Register erneut laden
0543 AND #$FC   Bits 0 und 1 löschen
0545 ORA $4B    und mit den errechneten Bits verknüpfen
0547 STA $1C00  und speichern
054A LDX #$09   $09 ist Zähler für
054C STX $4B    die Warteschleife,
054E DEC $4B    um dem Tonkopf
0550 BNE $054E  genügend Zeit
0552 DEX        zu geben,
0553 BNE $054E  positioniert zu werden
0555 RTS        Rücksprung
```

Sie werden sich sicher fragen, wie es möglich ist, daß sich der Tonkopf mittels dieser Routine bewegt. Für diese Bewegung sind die Bits 0 und 1 des Control-Registers zuständig. Der Kopf bewegt sich durch systematische Veränderung dieser Bits, wenn zusätzlich der Laufwerksmotor (Bit 3) eingeschaltet ist. Durch eine Veränderung der beiden Bits in der Folge 00/01/10/11/00 bewegt sich der Kopf weiter nach innen, und durch die Folge 00/11/10/01/00 bewegt er sich nach außen.

Hier ein Beispiel zum besseren Verständnis. Sind die Bits 0 und 1 wie in unserem Beispiel gelöscht und wird dann das erste Bit gesetzt, was der Bit-Folge 01 entspricht, so bewegt sich der Kopf, wenn man ihm genug Zeit läßt, um eine Halbspur nach innen. Durch das Setzen beider Bits bewegt er sich nach außen. Wie Sie sicher festgestellt haben, wird dieses Programm nicht über die Job-Codes, sondern direkt gestartet und läuft folglich auch nicht als Interrupt-Programm. Zum Schluß noch ein Beispiel für das Starten eines Programms mittels des Job-Codes $E0.

Das folgende Programm fährt den Tonkopf auf den eingestellten Track und liest dort den ersten Blockheader, den es findet. Die gelesenen Daten werden aus dem GCR-Format in die normalen Bit-Werte gewandelt und in den Adressen $16 bis $1A gespeichert. Die Speicherung geschieht in der Reihenfolge:

```
$16 - Erste ID auf Diskette
$17 - Zweite ID auf Diskette
$18 - gelesener Track
$19 - gelesener Sektor
$1A - Prüfsumme über den Blockheader
```

Die GCR-Daten des Headers stehen ab $24.

```
0500 JMP $0503   Sprung für den ersten Durchlauf unbedeutend
0503 LDA #$19    Low-Byte der Einsprungadresse
0505 STA $0501   in den JMP-Befehl schreiben (JMP $0519)
0508 LDA #$01    Track, auf dem ausgeführt werden soll
050A STA $0A     in Speicher für Puffer zwei speichern
050C LDA #$00    Sektornummer (unerheblich)
050E STA $0B     speichern
0510 LDA #$E0    Job-Code $E0 (Programm im Puffer ausführen)
0512 STA $02     in Job-Speicher für Puffer zwei schreiben
0514 LDA $02     Job-Speicher lesen
0516 BMI $0514   verzweige wenn nicht beendet
0518 RTS         Rücksprung
```

Das auszuführende Interrupt-Programm

```
0519 LDA #$03    Einsprung wieder
051B STA $0501   normalisieren (JMP $0503)
051E LDX #$5A    90 Leseveruche
0520 STX $4B     im Zähler speichern
0522 LDX #$00    Zeiger auf 0 setzen
0524 LDA #$52    GCR-Codierung $08 (Header-Kennzeichen)
0526 STA $24     in Arbeitsspeicher speichern
0528 JSR $F556   auf SYNC warten
052B BVC $052B   auf BYTE-READY beim Lesen warten
052D CLV         BYTE-READY wieder löschen
052E LDA $1C01   gelesenes Byte vom Port holen
0531 CMP $24     mit gespeichertem Header vergleichen
0533 BNE $0548   verzweige, wenn kein Blockheader gefunden
0535 BVC $0535   sonst auf BYTE-READY warten
0537 CLV         Leitung rücksetzen
0538 LDA $1C01   gelesenes Byte holen
053B STA $25,x   und in Arbeitsspeicher schieben
053D INX         Zeiger erhöhen
053E CPX #$07    schon ganzen HEADER gelesen?
0540 BNE $0535   verzweige, wenn noch nicht alle Zeichen
0542 JSR $F497   GCR-BYTE in Bit-Form wandeln
0545 JMP $FD9E   Rücksprung aus dem Interrupt (ok)

0548 DEC $4B     Zähler für Fehlversuche verringern
054A BNE $0522   verzweige wenn weitere Versuche
054C LDA #$02    Fehlermeldung ($02=Blockheader nicht
054E JMP $F969   gefunden) ausgeben und Programm beenden
```

Das Programm wird ab $0500 gestartet und stellt den JMP $0503 auf JMP $0519 um. Daraufhin wird die Track-Nummer übergeben und der Job-Code $E0 in den Job-Speicher $02 (Pufffer 2 - $0500) übergeben. Dieser Code wird nun von der interrupt-gesteuerten Job-Schleife erkannt, die mit der Bearbeitung des Co-

des anfängt. Als erstes wird der Tonkopf auf den entsprechenden Track positioniert und das auszuführende Programm ab $0500 (für Puffer 2) gestartet. Da der erste Befehl jedoch JMP $0519 ist, wird der Programmteil, der vom Benutzer gestartet wurde, übersprungen und der JMP-Befehl wieder auf JMP $0503 zurückgesetzt. Das jetzt arbeitende Programm läuft interrupt-gesteuert. Das Hauptprogramm wartet bei $0514 bis $0516 darauf, daß eine Rückmeldung vom Interrupt-Programm ankommt. Das Hauptprogramm muß immer mit einem RTS abgeschlossen werden, da sonst keine Meldung an den Computer geht und die Floppy sich aufhängt.

Das Interrupt-Programm darf nicht mit einem RTS abgeschlossen werden. Am Ende dieses Programms muß wieder in die Job-Schleife zurückgekehrt werden, welche die im Akku enthaltene Rückmeldung an den Job-Speicher übergibt. Nachdem das Hauptprogramm die Rückmeldung erhält, wird es mit dem RTS beendet. Mit $0545 JMP $FD9E wird eine $01 in den Akku geladen und danach ebenfalls mit JMP $F969 in die Job-Schleife gesprungen. Einer kleinen Erklärung bedarf es wohl noch bei

```
$0524 LDA #$52
```

Die $52 stellt die ersten 8 Bits einer im GCR-Format geschriebenen $08 dar, die bekanntlich für das Kennzeichen des Anfangs eines Blockheaders steht. Normalerweise kann nach dem Auffinden einer SYNC-Markierung nur eine $08 (Blockheader) oder $07 (Datenheader) gefunden werden. Da die GCR-Daten der beiden Zahlen sich jedoch schon nach den ersten 8 Bits unterscheiden, reicht es aus, nur diese zu vergleichen.

Hex	Bin	GCR
7	00000111	01010101 11 = $55+%11
8	00001000	01010010 01 = $52+%01

Wenn wir schon von GCR-Codierung sprechen: Wie kann man Daten, die man auf Disk schreiben will, in das GCR-Format und umgekehrt wieder ins Binärsystem wandeln? Für diese Fälle gibt es einige Routinen im DOS der Floppy.

$F78F Diese Routine wandelt den gesamten aktiven Puffer ins GCR-Format. Da bei der Codierung von 4 Bits zu 5 Bits selbstverständlich mehr Platz benötigt wird, werden die ersten GCS-Daten in dem Ausweichpuffer vom $01BB bis $01FF geschrieben. Die restlichen Daten stehen im aktiven Puffer.

$F5F2 Die Routine wandelt die GCR-Daten, die im Ausweichpuffer und im aktuellen Puffer stehen, wieder in den Binärcode zurück und speichert die Werte im aktuellen Puffer.

$F934 Wandelt den Blockheader, dessen Werte in den Speicherstellen $16 bis $1A stehen, in GCR-Daten um und speichert sie ab $24.

$F497 Wandelt den gelesenen Blockheader in Binärcode um. Die GCR-Daten stehen hierbei ab $24 und die Binärwerte ab $16 (siehe Beispielprogramm).

Das sollte als Einführung in die Programmierung der Floppy reichen. Zum Abschluß noch eine Beschreibung der Register der VIA 2, die von großem Interesse für die Programmierung der Floppy ist.

Wie funktioniert die Formatroutine?

Jeder von Ihnen wird schon Erfahrung mit der Formatroutine der Floppy gemacht haben, doch hier wollen wir sie einmal nicht aus der Sicht des Anwenders, sondern aus der Sicht des Programmierers betrachten.

Die Formatroutine, die bei $FAC7 beginnt, wird von der Routine ab $C8C6 mit dem Job-Code $E0 aufgerufen. Dazu wird von dieser Routine in die Speicherstellen $0600 bis $0602 ein "JMP $FAC7" geschrieben und dieser mit dem $E0-Befehl gestartet.

Ab jetzt beginnt die eigentliche Formatroutine. Sie prüft die Speicherstelle $51, die den Track angibt, der gerade formatiert wird. Sollte der Wert dieser Speicherstelle $FF sein, so erkennt das Programm, daß die Formatierung gerade begonnen wurde, und führt einen BUMP (Anschlag des Kopfes) aus. Im folgenden

wird der Track genau vermessen, und aufgrund dieser Messung werden die Abstände zwischen den Sektoren festgelegt. Diese Messung ist auch der Grund dafür, daß das Formatieren einer Diskette so lange dauert. Die Sektoren werden nun mitsamt ihren Headern aufgetragen, womit auch schon die Formatierung eines Tracks beendet ist. Nun wird die aktuelle Track-Nummer mit 36 verglichen. Sollte sie kleiner sein, verzweigt das Programm zur Track-Positionierung, von wo aus wieder in die Job-Schleife gesprungen wird. Die Job-Schleife prüft erneut, ob ein Job-Befehl ausgeführt werden soll, und erkennt den $E0-Job, der sich immer noch in $03 (Job-Speicher für Puffer 3) befindet. Das führt dazu, daß das Formatierungsprogramm ab $FAC7 durch den JMP-Befehl ab $0600 erneut aufgerufen wird. Wenn man in die Formatroutine eingreifen will, reicht es aus, ab $0600 eine eigene Routine einzusetzen.

Das sollte als Einführung in die Programmierung der Floppy reichen. Viele weitere Erfahrungen und Anregungen können Sie durch häufiges und intensives Studium der DOS-Listings erhalten. Zum Abschluß noch die Register-Beschreibung der Drive-Control-Register.

Register-Beschreibung der VIA 2

```
$1C00 Drive-Control-Bus
      Bit 0 Steppermotorsteuerung
      Bit 1 Steppermotorsteuerung
      Bit 2 Laufwerksmotor;     Bit=0: Motor aus
      Bit 3 Laufwerk-LED;       Bit=0: LED aus
      Bit 4 Schreibschutz;      Bit=0: Lichtschranke unterbrochen
      Bit 5 Zur Einstellung der Bit-Rate zum Tonkopf (Speed-Flag)
      Bit 6 Bedeutung wie Bit 5
      Bit 7 SYNC-Signal beim Lesen gefunden; Bit=0: SYNC gefunden
```

```
$1C01 Port A Ein-/Ausgabe-Register - Datenbus zum R/W-Kopf
```

```
$1C02 Datenrichtungs-Register Port B
```

```
$1C03 Datenrichtungs-Register Port A
```

```
$1C04 Timer 1 (Low)
```

```
$1C05 Timer 1 (High)
```

$1C06 Timer 1 (Low Latch)
Beim Schreib- oder Lesezugriff bleibt der Wert des Timers unverändert. Der Wert wird in diesem Zwischenspeicher gespeichert und im FREE-RUNNING-MODE ($1C11 Bit 6=1) bei jedem Null-Durchgang in den Zähler automatisch übernommen.

$1C07 Timer 1 (High Latch)
Siehe $1C06

$1C08 Timer 2 (Low)

$1C09 Timer 2 (High)

$1C0A Schiebe-Register

$1C0B Hilfssteuer-Register
Bit 0 PA (Latch-Enable-Disable)
Bit 1 PB (0=Disable 1=Enable Latching)
Bit 2 Schiebesteuer-Bit
Bit 3 Schiebesteuer-Bit
Bit 4 Schiebesteuer-Bit
Bit 5 T2 Timercontrol (0=Zeitlicher Interrupt 1=Abwärtszählen mit Signal am Anschuß PB6)
Bit 6 T1 Timercontrol
Bit 7 T1 Timercontrol

$1C0C Peripheriesteuer-Register (PCR)
Bit 0 CA1 Interrupt-Steuerung (0=Negative Flanke des Signals/1=Positive Flanke)
Bit 1 CA2 Control
Bit 2 CA2 Control
Bit 3 CA2 Control
Bit 4 CB1 Interrupt-Steuerung siehe Bit 0
Bit 5 CB2 Control
Bit 6 CB2 Control
Bit 7 CB2 Control

$1C0D Interrupt-Flag-Register
Dieses Register signalisiert das Eintreffen von Ereignissen durch Setzen des entsprechenden Bits dieses Registers.
Bit 0 Aktive Flanke an CA2
Bit 1 Aktive Flanke an CA1
Bit 2 acht Schiebeimpulse von SR ($1C0A)
Bit 3 Aktive Flanke an CB2
Bit 4 Aktive Flanke an CB1
Bit 5 Timer-Unterlauf von Timer 2
Bit 6 Timer-Unterlauf von Timer 1
Bit 7 Wird gesetzt, wenn eines der Bits sowohl in diesem Register als auch in $1C0E gesetzt ist.

```
$1C0E Interrupt-Enable-Register
      Die Bits dieses Registers korrespondieren mit den
      Bits aus $1C0D. Ist in diesem Register ein Bit
      gesetzt und der entsprechende Zustand, der in $1C0D
      angezeigt wird, erfüllt, wird ein IRQ ausgeführt.
```

```
$1C0F Daten-Register A
      Dieses Register spiegelt den Inhalt aus $1C00, jedoch
      ohne Handshake-Betrieb.
```

Die Formatroutine

Durch Änderung der Formatroutine lassen sich einige gut zu gebrauchende Effekte erzielen. Es ist möglich, einen einzelnen Track oder die Spuren 36 bis 41 zu formatieren sowie sämtliche Blockheader-Parameter zu verändern.

5.8.3 Formatieren eines einzelnen Tracks

Weiter vorne wurde schon allgemein auf die Arbeitsweise der Formatroutine eingegangen. Dort haben Sie erfahren, daß die Formatroutine vor der Formatierung eines Tracks jedesmal neu über den "JMP $FAC7" bei $0600 aufgerufen wird. An dieser Stelle greifen wir nun ein und zwingen die Routine, nur einen Track zu formatieren. Für dieses Vorhaben reicht das folgende kleine Programm, das mit dem Job-Code E0 im Job-Speicher $03 aufgerufen werden muß.

```
0600 JMP $0603   der JMP wird vom Programm noch verändert
0603 LDA $0C     Nummer des zu formatierenden Tracks holen
0605 STA $51     und übergeben
0607 LDA #$0F    JMP-Befehl auf $060F zeigen lassen
0609 STA $0601   um beim erneuten Aufruf die Formatierung
                 zu beenden
060C JMP $FAC7   zur Formatroutine springen
060F JMP $FD9E   beim zweiten Aufruf Programm beenden
```

Hier wird das Programm mit einem B-E-Befehl gestartet

```
0612 LDA #$01    Nummer des zu formatierenden Tracks laden
0614 STA $0C     und an den Job übergeben
0616 LDA #$E0    Job-Befehl E0 (Programm aufführen) laden
0618 STA $03     und in den Job-Speicher schreiben
061A RTS         Rücksprung
```

Als nächstes folgt ein BASIC-Programm, das es erlaubt, die entsprechende Nummer des zu formatierenden Tracks einzugeben. Zu Beginn des Programms wird initialisiert, damit die Formatroutine den Track auch mit der richtigen ID formatiert. Sollte dies nicht der Fall sein, läßt sich ein Block auf diesem Track nicht mehr ohne eigene Software lesen. Ein so "zerstörter" Track ist jedoch für Kopierprogramme des heutigen Standards kein Problem. Er kann lediglich genutzt werden, um Knackern den Zugriff auf den Block, auf dem die Kopierschutzabfrage steht, zu erschweren. Jetzt aber das angekündigte Programm.

```
OPEN1,8,15,"I"
20 READ X:IF X=-1THEN 100
30 PRINT#1,"M-W"CHR$(N)CHR$(6)CHR$(1)CHR$(X)
40 N=N+1:GOTO 20
100 INPUT "WELCHER TRACK";T
110 IF T >35 OR T<1 THEN100
120 PRINT#1,"M-W"CHR$(19)CHR$(6)CHR$(1)CHR$(T)
130 PRINT#1,"M-E"CHR$(18)CHR$(6)
140 PRINT#1,"M-R"CHR$(3)CHR$(0)CHR$(1)
150 GET#1,A$:A=ASC (A$+CHR$(0))
160 IF A>127 THEN130
170 IFA =1 THENPRINT"FORMATING OK":END
180 PRINT"FORMAT ERROR":END
302 DATA 76,  3,  6,165, 12,133, 81,169, 15,141,  1,  6, 76,199,250, 76,
158,253
304 DATA169,  1,133, 12,169,224,133,  3, 96,  -1
```

Wie schon gesagt, wird als erstes initialisiert. Daraufhin werden die Daten in die Floppy ab $0600 geschickt. Nachdem die Track-Nummer eingegeben wurde, wird sie ebenfalls zur Floppy geschickt und das Programm ab $0612 mit dem M-E-Befehl gestartet. Im Anschluß daran wird auf die Rückmeldung im Job-Speicher $03 gewartet und nach ihrem Erhalt die entsprechende Meldung ausgegeben.

Es ist nicht möglich, mit diesem Programm die Spuren oberhalb von 35 zu formatieren, was nicht von der Abfrage in Zeile 110, sondern von dem DOS der Floppy abhängt.

Will man Track 18 formatieren, so stößt man auf etwas größere Probleme. War dieser vor der Formatierung zerstört, so daß es nicht möglich ist zu initialisieren, muß vor der Formatierung die ID "per Hand" angegeben werden. Für diesen Zweck müssen Sie

mit dem M-W-Befehl die ID in die Floppy-Speicherstellen $12 (18) und $13 (19) schreiben. Sie werden sich vielleicht fragen, wie Sie die ID Ihrer Diskette erfahren sollen, falls Sie das Directory nicht mehr laden können. Für diesen Fall können Sie das weiter vorne beschriebene Programm verwenden, das den nächsten Blockheader, den es findet, liest und decodiert. Die ID des Blockheaders steht dann in Speicherstelle $16 (22) und $17 (23), von wo Sie sie ohne weiteres auslesen können.

Nach dem Formatieren des Tracks 18 kann man jedoch nicht gleich Programme auf der Diskette speichern. Zuvor müssen noch die BAM und das Directory erstellt und auf Diskette gespeichert werden. Dies erledigt für uns zum größten Teil eine Routine im DOS der Floppy. Wir müssen ihr lediglich den Namen unserer Diskette übergeben. Es ist nicht möglich, die Diskette "soft" zu formatieren (Formatierungsbefehl ohne ID-Angabe), da dieser Befehl nur angewendet werden kann, wenn die BAM schon vorher vorhanden war. Das folgende Programm wird in der Floppy ab $0400 mit einem M-E-Befehl gestartet. Die Daten für den Diskettennamen müssen in diesem Fall ab $0420 im Speicher stehen und mit Komma abgeschlossen werden.

```
0400 LDX #$10      16 Buchstaben des Namens
0402 LDA $0420,x   vom $0420 in den
0405 STA $0200,x   Befehlspuffer schieben
0408 DEX           Zähler verringern
0409 BPL $0402     verzweige, wenn noch nicht fertig
040B LDA #$10      16 Buchstaben für den Namen
040D STA $0274     in Puffer zulassen
0410 JMP $EE40     Befehl ausführen, Rücksprung
```

Das folgende BASIC-Programm erledigt alle Aufgaben für Sie. Der Name wird der Floppy übergeben, und daraufhin wird das Programm ausgeführt.

```
0 OPEN1,8,15,"I"
20 READ X:IF X=-1THEN 100
30 PRINT#1,"M-W"CHR$(N)CHR$(4)CHR$(1)CHR$(X)
40 N=N+1:GOTO 20
100 INPUT "DISKNAME";N$:N$=N$+","
110 FORN=0TO LEN(N$)-1
120 PRINT#1,"M-W"CHR$(32+N)CHR$(4)CHR$(1)MID$(N$,N+1,1): NEXT
130 PRINT#1,"M-E"CHR$(0)CHR$(4)
```

```
302 DATA162, 16,189, 32,  4,157,  0,  2,202, 16,247,169, 16,141,116,  2,
 76, 64
304 DATA238,-1
```

Das Programm fügt selbständig das Komma am Ende des Namens an. Mit diesen Programmen ist es Ihnen also auch möglich, einen einzelnen Track zu formatieren, solange er nicht oberhalb von Spur 35 liegt.

Formatieren der Spuren 36 bis 41

Es ist der Floppy ohne weiteres möglich, ihren R/W-Kopf auf die Spuren oberhalb der Spur 35 zu positionieren. Wollen Sie diese Spuren für die Ablage von Daten verwenden, so stoßen Sie noch auf ein Problem. Sie können den R/W-Kopf zwar ohne Probleme auf die Spuren oberhalb von 35 fahren, dort jedoch nicht ohne weiteres lesen. Denn die Tabelle, aus der sich das DOS die Anzahl der Sektoren pro Track und den Speed der entsprechenden Spur holt, gilt nur für die Spuren unterhalb von 36. Wir müssen deswegen sowohl beim Lesen als auch beim Schreiben den Speed und die Anzahl der Sektoren selber übergeben. Damit steht dem Formatieren der Spuren 36 bis 41 nichts mehr im Wege. Zu beachten ist lediglich, daß die Formatroutine nach der Formatierung eines Tracks erkennt, daß sie sich oberhalb von Spur 35 befindet und den Vorgang daraufhin abbricht. Wir müssen somit diese Routine vor der Formatierung eines weiteren Tracks erneut starten, was bei der Formatierung unterhalb von 36 nicht gemacht werden mußte. Im folgenden zeigen wir das Programm, das diese Spuren formatiert. Es wird in der Floppy mit dem M-E-Befehl ab $0415 gestartet.

```
0400 LDA #$11    Anzahl der Sektoren für die
0402 STA $43     Spuren über 35 übergeben
0404 LDA $1C00   Control-Port laden
0407 AND #$9F    und Speed für diese Tracks
0409 ORA #$00    einstellen (kann hier geändert werden)
040B STA $1C00   Wert speichern
040E LDA $08     Track-Nummer laden
0410 STA $51     und die Nummer übergeben
0412 JMP $FAC7   Formatroutine anspringen
```

Das Programm wird hier gestartet.

```
0415 JSR $D042   initialisieren
0418 LDA #$24    Track, bei dem die Formatierung beginnt,
041A STA $37     laden $24 (36) und zwischenspeichern
041C STA $08     Track-Nummer an Job übergeben
041E LDA #$E0    Job-Code E0 für Programm ausführen
0420 STA $01     in Job-Speicher schreiben
0422 LDA $01     Rückmeldung erwarten
0424 BMI $0422   verzweige, wenn Job nicht abgearbeitet
0426 CMP #$02    Rückmeldung auf Fehler prüfen
0428 BCS $0432   verzweige, wenn Fehler aufgetreten
042A INC $37     Track-Nummer erhöhen
042C LDA $37     und vergleichen,
042E CMP #$2A    ob Spur 41 schon überschritten
0430 BNE $041A   wenn nicht, nächsten Track formatieren
0432 RTS         Rücksprung
```

Im folgenden ein BASIC-Programm, das Ihnen die Eingabe des Anfangs- und des End-Tracks erlaubt. Der End-Track sollte nicht höher als Spur 42 liegen, da der R/W-Kopf sonst anschlägt und sich dadurch das Laufwerk dejustieren kann. Track 42 schaffen die meisten Floppys noch gerade zu formatieren, sofern eine qualitativ hochwertige Diskette verwendet wird. Diesen Track erreicht bis heute kein von uns bekanntes Kopierprogramm.

```
10 OPEN 1,8,15,"I"
20 READ X:IF X=-1THEN 100
25 SU=SU+X
30 PRINT#1,"M-W"CHR$(N)CHR$(4)CHR$(1)CHR$(X)
40 N=N+1:GOTO 20
100 IF SU <> 5381 THENPRINT"FEHLER IN DATAS":STOP
102 INPUT "STATRTRACK >35 ";S
105 INPUT "ENDTRACK <42 ";E
120 PRINT#1,"M-W"CHR$(25)CHR$(4)CHR$(1)CHR$(S)
125 PRINT#1,"M-W"CHR$(47)CHR$(4)CHR$(1)CHR$(E+1)
130 PRINT#1,"M-E"CHR$(24)CHR$(4)
140 PRINT#1,"M-R"CHR$(1)CHR$(0)CHR$(1)
150 GET#1,A$:A=ASC (A$+CHR$(0))
160 IF A>127 THEN140
170 IFA =1 THENPRINT"FORMATING OK":END
180 PRINT"FORMAT ERROR":END
302 DATA169, 17,133, 67,173, 0, 28, 41,159, 9, 0,141, 0, 28,165, 8,133,
81
304 DATA 76,199,250, 32, 66,208,169, 36,133, 55,133, 8,169,224,133, 1,16
5, 1
306 DATA 48,252,201, 2,176, 8,230, 55,165, 55,201, 42,208,232, 96, -1
```

Wir können uns jetzt mit Hilfe dieses Programms die Spuren 36 bis 41 formatieren und sie somit für die Speicherung von Daten

vorbereiten. Doch die Speicherung und das Laden von Daten auf diesen Spuren ist nicht so leicht wie auf den normal verwendeten Tracks. Wie schon gesagt, kann das DOS für diese "illegalen" Spuren den Speed und die Sektorenanzahl nicht richtig bestimmen. Aus diesem Grund kann man weder mit den USER-Befehlen U1 (Block laden) und U2 (Block speichern) noch mit den Job-Codes 80 (Block laden) sowie 90 (Block speichern) arbeiten. Die einzige Möglichkeit, die uns bleibt, ist, über den Job-Code E0 ein Programm aufzurufen, das den Speed und die Anzahl der Sektoren setzt und danach die Block-Lese- oder Schreibroutine anspringt. Das Programm, das einen Block liest, sieht dann folgendermaßen aus:

```
0600 LDA $1C00    Control-Port laden
0603 AND #$9F     Bits für Speed löschen
0605 ORA #$00     und mit neuem Speed verknüpfen
0607 STA $1C00    Wert wieder speichern
060A LDA #$11     Anzahl der Sektoren laden
060C STA $43      und übergeben
060E LDA #$05     High-Byte der Pufferadresse, in den
0610 STA $31      geladen werden soll, angeben
0612 JMP $F4D1    zur Laderoutine springen
```

Das Programm wird hier gestartet.

```
0615 LDA #$24     zu ladenden Track für
0617 STA $0C      den Job übergeben
0619 LDA #$00     Sektor-Nummer an
061B STA $0D      den Job übergeben
061D LDA #$E0     Job-Code für Programm ausführen in den
061F STA $03      Job-Speicher schreiben
0621 LDA $03      Rückmeldung abwarten
0623 BMI $0621    verzweige, wenn noch nicht fertig
0625 RTS          Rücksprung
```

Das Programm wird ab $0615 mit dem M-E-Befehl gestartet und lädt den angegebenen Sektor in Puffer 2 ($0500). Auch hier müssen vor dem Laden des Blocks der Speed und die Anzahl der Sektoren übergeben werden, worauf die Routine zum Lesen eines Blocks angesprungen wird. Wenn Sie einen Block auf die oberen Spuren sichern wollen, so reicht es aus, den JMP $F4D1 bei Adresse $0612 in ein JMP $F575 zu ändern. Das nachfolgende BASIC-Programm erlaubt Ihnen die Eingabe des Tracks und des Sektors, von dem Sie laden wollen.

```
10 OPEN1,8,15,"I"
20 READ X:IF X=-1THEN 100
25 SU=SU+X
30 PRINT#1,"M-W"CHR$(N)CHR$(6)CHR$(1)CHR$(X)
40 N=N+1:GOTO 20
100 IFSU <> 3608 THEN PRINT"ERROR IN DATAS":STOP
105 INPUT "WELCHER TRACK";T
110 INPUT "WELCHER SEKTOR";S
120 PRINT#1,"M-W"CHR$(22)CHR$(6)CHR$(1)CHR$(T)
125 PRINT#1,"M-W"CHR$(26)CHR$(6)CHR$(1)CHR$(S)
130 PRINT#1,"M-E"CHR$(21)CHR$(6)
135 FORN=1TO 500:NEXT
140 PRINT#1,"M-R"CHR$(3)CHR$(0)CHR$(1)
150 GET#1,A$:A=ASC (A$+CHR$(0))
160 IF A>127 THEN130
170 IFA =1 THENPRINT"OK":END
180 PRINT"ERROR":END
302 DATA173, 0, 28, 41,159, 9, 0,141, 0, 28,169, 17,133, 67,169, 5,133,
49
304 DATA 76,209,244,169, 36,133, 12,169, 0,133, 13,169,224,133, 3,165,
3, 48
306 DATA252, 96, -1
```

Der gelesene Block wird, wie schon gesagt, in Puffer 2 ($0500) geschrieben, von wo aus er in den Computer geholt werden kann. Um mit diesem Programm einen Block zu schreiben, ist es notwendig, die Zeile 304 und die Prüfsumme über die Daten zu ändern. Anstatt 304 DATA 76,209,244 ... muß hier 304 DATA 76,117,245 ... stehen. Die Prüfsumme in Zeile 100 muß dann 3517 anstelle von 3608 lauten. Die Daten, die auf den eingegebenen Track und Sektor geschrieben werden, werden natürlich auch wieder aus Puffer 2 ($0500) geholt. Es ist nicht möglich, mit diesem Programm Blöcke auf den Tracks unterhalb von Track 31 zu schreiben, da diese Tracks mit einem anderen Speed beschrieben werden.

5.8.4 Doppelte Spuren

Ein Kopierschutzsystem, daß sich weniger durch seine Kompliziertheit als durch seine Raffiniertheit auszeichnet, soll nun beschrieben werden. Es ist schwer auszumachen, weil keine Fehler auf der Diskette zu erkennen sind. Bei diesem Kopierschutz handelt es sich auch, wie Sie sich sicher denken können, um eine Änderung der Formatroutine. Der Gedanke, der hinter die-

sem Kopierschutz steckt, ist recht einfach. Die unteren Spuren werden doppelt angelegt. Das heißt, daß die Tracks auf Diskette anstelle der gewohnten Reihenfolge Track 1, Track 2, Track 3 usw. die Folge Track 1, Track 2, Track 1, Track 2, Track 3, Track 4, Track 5 usw. aufweisen. Sie erkennen, daß der Track 1 und Track 2 doppelt auf Diskette vorhanden sind.

Sie werden sich vielleicht fragen, was das in Hinblick auf Kopierschutz bedeutet. Die Antwort ist einfach, wenn man sich daran erinnert, wie das DOS auf die einzelnen Tracks zugreift. Als erstes wird "nachgesehen", auf welchem Track sich der R/W-Kopf gerade befindet. Danach wird die Differenz zwischen der jetzigen und der zu ereichenden Spur errechnet und der R/W-Kopf um die entsprechende Anzahl von Spuren verschoben. Der Kopf erreicht nach diesem Prinzip somit nie auf normalem Wege die gedoppelten, unterhalb der echten liegenden Tracks 1 und 2. Nach genau diesem Prinzip arbeitet auch die Positionierung des R/W-Kopfes bei den Kopierprogrammen. Für diese ist es nicht erkennbar, daß unterhalb des gefundenen Track 1 sich noch 1, 2 oder mehr Tracks befinden. Es gibt keine Probleme beim Arbeiten mit der Diskette, auch wenn der Kopf einmal einen "Bump" (Anschlag des Kopfes) machen sollte, denn er fährt zu der vor dem Anschlag errechneten Spur, erkennt, daß er sich noch um einige Spuren zu tief befindet, und fährt den Kopf an die richtige Position.

Wie schon anfangs gesagt, ist es sehr einfach, eine Diskette derart zu formatieren. Man muß nur vorher den Kopf um die eingestellte Anzahl von Spuren nach innen verschieben. Von dieser Verschiebung merkt das DOS nichts und "denkt", es würde den Kopf auf Spur 1 fahren. Statt dessen beginnt es jedoch die Formatierung einige Spuren höher.

Nach der ausführlichen Erklärung des Prinzips wollen wir zur Praxis übergehen und ein solches Programm ausprobieren. Das folgende Maschinenspracheprogramm, das wie immer in der Floppy gestartet wird, erfüllt diesen Zweck. Es wird ab $0450 mit einen M-E-Befehl gestartet

```
0400 JMP $0403    für ersten Aufruf belanglos
0403 LDA #$17     Low-Byte für die Änderung des JMP
0405 STA $0401    Low-Byte des JMP auf $0417 ändern
0408 LDA #$04     Anzahl der Halbspuren, die verschoben
040A STA $37      werden, zwischenspeichern
040C JSR $041A    Kopf um eine Halbspur verschieben
040F DEC $37      Zähler verringern
0411 BNE $040C    verzweige, wenn nicht genug verschoben
0413 LDA #$01     Nummer des Tracks, ab dem formatiert
0415 STA $51      werden soll, übergeben
0417 JMP $FAC7    zur Formatroutine springen
```

Kopf-Verschieberoutine

```
041A LDX $1C00    Control-Port laden
041D INX          erhöhen für Kopfbewegung nach innen
041E TXA          Wert in Akku schieben
041F AND #$03     erste zwei Bits isoliern
0421 STA $4B      und zwischenspeichern
0423 LDA $1C00    Control-Port laden
0426 AND #$FC     Bit 0 und 1 löschen
0428 ORA $4B      mit den errechneten Bits verknüpfen
042A STA $1C00    und Speichern
042D LDX #$00     Zähler für Warteschleife setzen, um
042F LDY #$05     dem Kopf genügend Zeit zu lassen
0431 DEX          positioniert zu werden
0432 BNE $0431    verzweige, wenn Low-Byte nicht abgelaufen
0434 DEY          High-Byte verringern
0435 BNE $0431    springe, wenn High-Byte nicht abgelaufen
0437 RTS          Rücksprung
```

Das Programm wird hier gestartet.

```
0438 LDA #$12     Track 18 für Job
043A STA $08      übergeben
043C LDA #$E0     Job-Code E0 laden
043E STA $01      und in Job-Puffer schreiben
0440 LDA $01      Rückmeldung holen
0442 BMI $0440    verzweige, wenn noch nicht fertig
0444 CMP #$02     Rückmeldung auf Fehler prüfen
0446 BCS $0459    verzweige, wenn Fehler erkannt
0448 LDX #$10     Disknamen für BAM 16 Buchstaben erlauben
044A STX $0274    Anzahl übergeben
044D LDA $0480,X Namen von $0480 in
0450 STA $0200,X den Befehls-Puffer schreiben
0453 DEX          Zähler verringern
0454 BPL $044D    verzweige, wenn nicht alle Buchstaben
0456 JMP $EE40    BAM schreiben, Rücksprung
0459 RTS          Rücksprung bei Fehler
```

Im Anschluß an dieses Programm folgt ein BASIC-Programm, das uns die Eingabe der Anzahl der Tracks ermöglicht, die bei der Formatierung doppelt angelegt werden. Um diese Spezialformatierung nicht zu auffällig zu machen, sollte man nicht mehr als zwei Tracks höher mit der Formatierung beginnen. Damit dieses Programm richtig arbeitet, müssen die zu behandelnden Disketten schon fehlerfrei formatiert gewesen sein. Die neue Diskette wird mit der gleichen ID formatiert, mit der sie zuvor formatiert wurde. Doch nun zu dem angekündigten BASIC-Programm, das das Maschinenspracheprogramm als erstes zur Floppy schickt, um es nach der Eingabe der Parameter dort zu starten.

```
10 OPEN1,8,15,"I"
20 READ X:IF X=-1THEN 100
25 SU=SU+X
30 PRINT#1,"M-W"CHR$(N)CHR$(4)CHR$(1)CHR$(X)
40 N=N+1:GOTO 20
100 IF SU <> 9284 THENPRINT"FEHLER IN DATAS":STOP
102 INPUT "WIEVIELE TRACKS VERSCHIEBEN <8 : ";AN
104 INPUT "DISKNAME";N$:N$=N$+","
106 FORN=OTO LEN(N$)-1
108 PRINT#1,"M-W"CHR$(128+N)CHR$(4)CHR$(1)MID$(N$,N+1,1):   NEXT
120 PRINT#1,"M-W"CHR$(9)CHR$(4)CHR$(1)CHR$(AN*2)
130 PRINT#1,"M-E"CHR$(56)CHR$(4)
140 PRINT#1,"M-R"CHR$(1)CHR$(0)CHR$(1)
150 GET#1,A$:A=ASC (A$+CHR$(0))
160 IF A>127 THEN140
170 IFA =1 THENPRINT"FORMATING OK":END
180 PRINT"FORMAT ERROR":END
402 DATA 76, 3, 4,169, 23,141, 1, 4,169, 4,133, 55, 32, 26, 4,198, 55,20
8
404 DATA249,169, 1,133, 81, 76,199,250,174, 0, 28,232,138, 41, 3,133, 75
,173
406 DATA  0, 28, 41,252,  5, 75,141, 0, 28,162, 0,160, 5,202,208,253,136
,208
408 DATA250, 96,169, 18,133, 8,169,224,133, 1,165, 1, 48,252,201, 2,176,
 17
410 DATA162, 16,142,116, 2,189,128, 4,157, 0, 2,202, 16,247, 76, 64,238,
 96
412 DATA -1
```

Nachdem wir uns mit dem Auftragen des Kopierschutzes beschäftigt haben, wollen wir uns um die Abfrage desselben kümmern. Natürlich kann man nicht einfach mittels eines U1-Befehls der Floppy mitteilen, daß man gedenkt, einen Block auf Track 1 zu lesen, der unterhalb des vermuteten Track 1 liegt.

Um dieses zu können, ist es wieder notwendig, mit einem Maschinenspracheprogramm in der Floppy zu arbeiten. Dieses Programm fährt den R/W-Kopf auf den "normalen" Track 1, verschiebt mit Hilfe der schon verwendeten Routine den Kopf der Floppy um eine Spur nach außen und sucht erneut die Spur 1. Das erneute Suchen ist wichtig, da das DOS "denkt", es würde sich noch auf Track 1 befinden. Würde man einen anderen Track suchen lassen, würde der Kopf auf diese Spur fahren, die jedoch nicht unserer doppelten entspricht. Sucht man jedoch die Spur 1, auf der sich der Kopf vermeintlich befindet, so stellt das DOS fest, sofern mehr als eine Spur höher formatiert wurde, daß der Kopf sich auf einer höheren Spur befindet, und fährt den Kopf noch weiter nach unten, bis er die doppelte Spur 1 findet und sie als die richtige erkennt.

Nach dieser Prozedur kann man ohne Probleme die Blöcke auf diesen doppelten Spuren mit dem U1-Befehl lesen oder schreiben. Das folgende Maschinenspracheprogramm leistet das gerade Besprochene. Es wird ab $0627 in der Floppy gestartet.

```
0600 JSR $0609    R/W-Kopf um eine
0603 JSR $0609    Spur verschieben (2 mal eine Halbspur)
0606 JMP $FD9E    Rücksprung

0609 LDX $1C00    Control-Register laden
060C DEX          verringern, um Kopf nach außen zu fahren
060D TXA          Wert in Akku schieben
060E AND #$03     und die ersten zwei Bits isolieren
0610 STA $4B      Wert zwischenspeichern
0612 LDA $1C00    Control-Register laden
0615 AND #$FC     Bit 0 und 1 löschen
0617 ORA $4B      mit gespeichertem Wert verknüpfen
0619 STA $1C00    und Bewegung ausführen
061C LDX #$00     Werte für die
061E LDY #$05     Zeitschleife laden
0620 DEX          Zähler verringern
0621 BNE $0620    verzweige, wenn nicht abgelaufen
0623 DEY          High-Byte des Zählers verringern
0624 BNE $0620    verzweige, wenn nicht abgelaufen
0626 RTS          Rücksprung
```

Das Programm wird hier gestartet.

```
0627 JSR $D042   Disk initialisieren
062A LDA #$01    Track auf den der Kopf gefahren wird
062C STA $0C     übergeben
062E LDA #$E0    Job-Code E0 laden
0630 STA $03     und in Job-Speicher schreiben
0632 LDA $03     Rückmeldung erwarten
0634 BMI $0632   verzweige, wenn noch keine Rückmeldung
0636 LDA #$01    Nummer für doppelten Track 1 laden
0638 STA $0A     und übergeben
063A LDA #$00    Sektor 0 laden
063C STA $0B     und übergeben
063E LDA #$80    Job-Code 80 (Block lesen) laden
0640 STA $02     und übergeben
0642 LDA $02     auf Rückmeldung warten
0644 BMI $0642   verzweige, wenn keine Rückmeldung
0646 RTS         Rücksprung
```

Sie können anhand des Programms das Prinzip noch einmal nachvollziehen. Das Programm lädt Block 1 des doppelten (unteren) Tracks automatisch in Puffer 2 ($0500). Hier nun der dazugehörige BASIC-Loader.

```
OPEN1,8,15
20 READ X:IF X=-1THEN 100
30 SU=SU+X:PRINT#1,"M-W"CHR$(N)CHR$(6)CHR$(1)CHR$(X)
40 N=N+1:GOTO 20
100 IF SU<>7037THENPRINT"FEHLER IN DATAS":STOP
130 PRINT#1,"M-E"CHR$(39)CHR$(6)
302 DATA 32, 9, 6, 32, 9,  6, 76,158,253,174, 0, 28,202,138, 41,  3,133,
 75
304 DATA173, 0, 28, 41,252,  5, 75,141, 0, 28,162, 0,160, 5,202,208,253,
136
306 DATA208,250, 96, 32, 66,208,169, 1,133, 12,169,224,133, 3,165, 3, 48
,252
308 DATA169, 1,133, 10,169, 0,133, 11,169,128,133, 2,165, 2, 48,252, 96,
 -1
```

Wie schon erwähnt, können Sie nach dem Aufruf des Programms mit den normalen Block-Lese- und -Schreibbefehlen auf die Blöcke der unteren Tracks zugreifen. Wollen Sie nach Beendigung der Kopierschutzabfrage wieder auf die "richtigen" Tracks zugreifen, reicht es aus, die Diskette zu initialisieren oder einen nicht doppelt vorhandenen Track zu lesen.

Änderung der Header-Parameter: Read-Errors

Ein weiteres sehr sicheres Kopierschutzsystem läßt sich auch mit einer anderen Änderung der Formatroutine erreichen. In diesem Fall handelt es sich um die Änderung der Blockheader-Parameter. Wenn Sie schon etwas über die Änderung der Header-Parameter gelesen oder es schon selbst gemacht haben, werden Sie wahrscheinlich sehr skeptisch sein, ob man mit einem solchen Eingriff einen wirklich guten Kopierschutz erstellen kann. Aber lassen Sie sich überraschen.

Die einfache Blockheader-Manipulation dient nur dem Zweck, Daten für den Knacker schwerer zugänglich zu machen, was auch nicht zu verachten ist. Den Kopierprogrammen heutigen Standards ist es jedoch ein leichtes, diese Blöcke zu kopieren.

Bevor wir besprechen, wie man diese veränderten Parameter auf Diskette aufträgt, wollen wir Sie jedoch erst noch weiter in die Arbeitsweise der Formatroutine einführen, um die nachfolgenden Programme auch verstehen zu können.

Wir hatten schon gesagt, daß die Formatroutiene bei $FAC7 im Speicher der Floppy beginnt und auch dort aufgerufen wird. Nachdem der R/W-Kopf auf den angegebenen Track positioniert wurde, wird die Spur genau vermessen, um festzustellen, wie groß die Lücke zwischen den einzelnen Sektoren sein soll. Diese Zahl darf nicht kleiner als 4 sein. Nach der Errechnung wird die Länge der Lücke in $0626 zwischengespeichert. Nach dieser Speicherung beginnt der zweite Teil der Formatroutine ab $FC36. Hier werden jetzt die Blockheader für die zu formatierenden Sektoren vorbereitet. Zu diesem Zweck werden die Parameter aus der Zeropage der Floppy geholt. Der Blockheader-Code $08 steht in $39, die Track-Nummer wird aus der Adresse $51 geholt. Die ID steht in $12, $13, und die zwei Lückenwerte sind $0F. Sie werden direkt geladen und sind wie auch die Track-Nummer und die Sektornummer nicht veränderbar. Daraufhin wird die Prüfsumme für den Blockheader berechnet.

Diese 8 Daten werden ab $0300 hintereinander liegend für jeden Header abgelegt und gemeinsam in das GCR-Format umgerech-

net. Die Anzahl der Sektoren wird beim Erstellen der Blockheader in $0628 gespeichert. Vor der Umrechnung steht im Y-Register die Anzahl der Bytes, die zu den Blockheadern gehören.

Nach dieser Arbeit beginnt erst die eigentliche Formatierung, was bedeutet, daß die Blöcke mit ihren Blockheadern und alle SYNC-Markierungen auf Diskette geschrieben werden. Nach der Formatierung des Tracks wird die sich in $51 befindende Track-Nummer überprüft. Sollte sie höher oder gleich $24 (56) sein, wird die Formatierung beendet. Soweit die nähere Erläuterung der Formatroutine, die jedoch zum Verständnis des folgenden Programms nötig ist.

Um die Header-Parameter zu ändern, wollen wir diese nicht von der Formatroutine erstellen lassen, sondern sie selbst im Computer erstellen und dann an die richtige Stelle in der Floppy schreiben. Natürlich dürfen wir dann die Formatroutine nicht von Anfang an starten, sondern erst dort einspringen, wo die Routine die Header-Parameter in das GCR-Format wandelt. Um die Formatroutine dort starten zu können, müssen auch die richtigen Parameter an die Routine übergeben werden, die sonst bis zu diesem Punkt von der Formatroutine selbst errechnet worden wären, wie beispielsweise die Länge der Lücke zwischen den Sektoren.

Alle diese Aufgaben erledigt für Sie das folgende etwas längere Programm. Mit ihm ist es auf komfortable Weise möglich sämtliche Blockheader-Parameter zu ändern und somit einzelne Sektoren vor dem Zugriff anderer zu schützen oder einen guten Kopierschutz aufzutragen.

```
10 A=49152
20 READ X:IF X=-1 THEN40
30 POKEA,X:A=A+1:SU=SU+X:GOTO20
40 IF SU <> 17072 THENPRINT"FEHLER IN DATAS":STOP
50 PO=29:PRINTCHR$(147):PRINT:PRINT:PRINT
60 PRINT"    SPEZIALFORMATIERUNG EINES TRACKS"
70 PRINT:PRINT:INPUT "    GEBEN SIE DEN TRACK AN";T
80 SE=21:L=9:SP=3
90 IF T>17 THENSE=19:L=9:SP=2
100 IF T>24 THENSE=18:L=10:SP=1
110 IF T>30 THENSE=17:L=11:SP=0
120 IF T<1 OR T>41 THEN PRINTCHR$(145)CHR$(145)CHR$(145);: GOTO70
```

```
130 PRINT:PRINT"    BITTE LEGEN SIE DIE DISKETTE EIN"
140 GET A$:IF A$=""THEN140
150 OPEN1,8,15,"I"
160 PRINT#1,"M-R"CHR$(18)CHR$(0)CHR$(2)
170 GET#1,I1$,I2$
180 PRINT:PRINT"    ID 1 DER DISKETTE IST ";ASC(I1$+CHR$(0))
190 PRINT:PRINT"    ID 2 DER DISKETTE IST ";ASC(I2$+CHR$(0))
200 PRINT:PRINT"    NEUE ID 1 FUER DEN TRACK  ";ASC (I1$+CHR$(0));:GOSUB
840
210 POKE249,EI
220 PRINT:PRINT"    NEUE ID 2 FUER DEN TRACK  ";ASC(I2$+CHR$(0));:GOSUB8
40
230 POKE250,EI
240 POKE248,T:POKE 49285,T
250 POKE251,SE:POKE 49257,SE
260 POKE 49264,L
270 POKE49249,SE*8
280 SYS49216
290 PRINT:PRINT"    HEADER-MANIPULATION (J/N)?"
300 GET A$:IFA$=""THEN300
310 IF A$="N"THEN630
320 PRINTCHR$(147);
330 PRINTCHR$(19)CHR$(17)CHR$(17);"  WELCHER HEADER 0 -";SE-1;:INPUTH
340 IF H<0 OR H> SE-1 THEN 330
350 X=49664+H*8
360 PO=20:PRINT:PRINT"  HEADER-POSITION ";H
370 PRINT:PRINT:PRINT"  HEADER-KENNZEICHEN  ";PEEK(X);:GOSUB
840:POKEX,EI
380 PRINT"  ERSTE  ID           ";PEEK(X+5);:GOSUB 840:POKEX+5,EI
390 PRINT"  ZWEITE ID           ";PEEK(X+4);:GOSUB 840:POKEX+4,EI
400 PRINT"  TRACK-NUMMER         ";PEEK(X+3);:GOSUB840:POKEX +3,EI
410 PRINT"  SEKTOR-NUMMER        ";PEEK(X+2);:GOSUB840:POKEX +2,EI
420 PRINT"  LUECKE 1            ";PEEK(X+6);:GOSUB840:POKEX +6,EI
430 PRINT"  LUECKE 2            ";PEEK(X+7);:GOSUB840:POKEX +7,EI
440 PR=PEEK(X+1)
450 POKE49152+83,H:SYS 49152+82
460 PRINT"  ECHTE  PRUEFSUMME  ";PEEK(X+1)
470 PRINT"  EIGENE PRUEFSUMME  ";PR;:GOSUB840:POKEX+1,EI
480 PRINTCHR$(17)CHR$(17);"  ";CHR$(18);"G";CHR$(146);"LEICHER   HEADER"
490 PRINT"  ";CHR$(18);"A";CHR$(146);"NDRER     HEADER"
500 PRINT"  ";CHR$(18);"T";CHR$(146);"AUSCHE    HEADER"
510 PRINT"  ";CHR$(18);"K";CHR$(146);"OPIERE    HEADER"
520 PRINT"  ";CHR$(18);"E";CHR$(146);"NDE, FORMATIEREN"
530 GET A$:IF A$=""THEN530
540 IF A$="A" THEN GOTO320
550 IF A$="G" THEN PRINTCHR$(147);:GOTO360
560 IF A$="E"THEN630
570 IF A$="T"THEN1020
580 IF A$="K"THEN880
590 GOTO530
600 IF H>SETHENH=0
610 IF H<0 THENH=SE
620 GOTO330
```

```
630 PRINTCHR$(147)
640 FORB=1 TO 5:PRINTCHR$(17):NEXT
645 PRINT" GEBEN SIE DEN SPEED DES TRACKS AN  ";SP;:PO=35:GOSUB840
650 IF EI <0 OR EI>3 THEN640
660 POKE49277,EI*32:PRINTCHR$(147);
670 FORN=0TO SE*8-1:PRINTCHR$(19);"    ";CHR$19);SE*8-1-N
680 PRINT#1,"M-W"CHR$(N)CHR$(3)CHR$(1)CHR$(PEEK(49664+N)) :NEXT
690 FORN=0TO 48:PRINTCHR$(19);"    ";CHR$19);48-N
700 PRINT#1,"M-W"CHR$(N)CHR$(4)CHR$(1)CHR$(PEEK(49248+N)) :NEXT
710 PRINT#1,"M-E"CHR$(36)CHR$(4)
720 PRINT#1,"M-R"CHR$(1)CHR$(0)CHR$(1)
730 GET#1,A$:IF ASC (A$+CHR$(0))>128THEN720
740 IF ASC (A$+CHR$(0)) >1THENPRINT"FORMAT ERROR":GOTO800
750 PRINT"FORMAT OK     "
760 PRINT:PRINT:PRINT "  WEITERE TRACKS    "
770 GET A$:IF A$="" THEN770
780 IF A$<>"N" THEN CLOSE1:GOTO50
790 CLOSE1:END
800 PRINT:PRINT:PRINT"  WEITERER VERSUCH"
810 GET A$:IF A$="" THEN810
820 IF A$<>"N" THEN PRINTCHR$(147):GOTO670
830 GOTO760
840 POKE211,PO:GOTO850
850 SYS58732:INPUT EI
860 IF EI <0 OR EI>255 THEN840
870 RETURN
880 PRINTCHR$(147)CHR$(17)CHR$(17);
885 PRINT" AUF ";CHR$(18);"E";CHR$(146);"INE ODER ";
887 PRINTCHR$(18);"A";CHR$(146);"LLE POSITIONEN KOPIEREN"
890 GET A$:IF A$=""THEN890
900 IF A$="E" THEN  970
910 IF A$ <>"A"THEN 890
920 FORN=0TO SE-1:Y=49664+N*8
930 FORM=0TO 7
940 POKEY+M,PEEK(X+M):NEXT
950 NEXT
960 GOTO480
970 PRINT:PRINT:PRINT" AUF WELCHE POSITION 0 -";SE-1;
980 PO=30:GOSUB840:IF EI>SE-1 THEN PRINT CHR$(147);:GOTO980
990 Y=49664+EI*8
1000 FORM=0TO 7
1010 POKEY+M,PEEK(X+M):NEXT:GOTO480
1020 PRINTCHR$(147)CHR$(17)CHR$(17)" MIT WELCHER POSITION TAUSCHEN 0 -";
SE-1
1030 PRINT:PRINT:PO=17:GOSUB840
1040 IF EI>SE-1 THENPRINTCHR$(147);:GOTO1030
1050 Y=49664+EI*8
1060 FORN=0 TO 7
1070 HE=PEEK(Y+N):POKEY+N,PEEK(X+N)
1080 POKEX+N,HE
1090 NEXT:GOTO480
1100 REM
```

```
1110 DATA169,8,153,0,194,200,200,165,247,153,0,194,200,165,248,153,0,194
,200
1020 DATA165,250,153,0,194,200,165,249,153,0,194,200,169,15,153,0,194,20
0
1130 DATA153,0,194,200,169,0,89,250,193,89,251,193,89,252,193,89,253,193
,153
1140 DATA249,193,96,0,0,0,0,0,169,0,133,247,160,0,32,0,192,230,247,165,2
47
1150 DATA197,251,144,245,96,169,0,10,10,10,24,105,8,168,76,41,192,0,0,16
0
1160 DATA136,162,0,169,35,133,81,169,17,133,67,141,40,6,169,11,141,38,6,
141
1170 DATA32,6,173,0,28,41,159,9,0,141,0,28,76,132,252,169,35,133,8,169,2
24
1180 DATA133,1,165,1,48,252,96,-1
```

Anleitung zum Programm

Wenn Sie das Programm starten, müssen Sie ein wenig Geduld aufbringen, denn die Daten in den DATA-Zeilen müssen erst in den Speicher gepoket werden. Daraufhin wird die eingelegte Diskette initialisiert und die entsprechende ID angegeben. Es besteht die Möglichkeit, die ID zu ändern. Wenn Sie keine Blockheader-Manipulation durchführen wollen, somit die entsprechende Frage verneinen, wird der angegebene Track fehlerlos formatiert. Zuvor müssen Sie jedoch noch den Speed des Tracks angeben. Soll er nicht vom normalen abweichen, brauchen Sie nur <Return> zu drücken.

Änderung der Header-Parameter

Als erstes werden Sie nach der Position des zu ändernden Headers gefragt. Die einzige Angabe, die einer Erklärung bedarf, ist die Angabe der Prüfsumme. Die unter "echte Prüfsumme" ausgegebene Zahl ist die aus den zuvor gemachten Angaben errechnete Prüfsumme. Unter "eigene Prüfsumme" wird diejenige angezeigt, die man selbst gewählt hat. Im Anschluß an diese Eingaben erscheint ein Menü, bei dem Sie unter folgenden Punkten auswählen können:

1. *Gleicher Header*
   erlaubt eine erneute Modifikation desselben Headers.
2. *Anderer Header*
   ermöglicht die Änderung eines weiteren Headers.

3. *Tausche Header*
   ermöglicht die Vertauschung von zwei Headern und somit eine Änderung der Reihenfolge der Sektoren auf einem Track, was auch einen recht guten Kopierschutz abgibt.
4. *Kopiere Header*
   erlaubt, einen Header zu duplizieren und dadurch zwei gleiche Sektoren auf einem Track zu erzeugen, was nicht von allen Kopierprogrammen korrekt kopiert werden kann. Rufen Sie diesen Menüpunkt auf, werden Sie gefragt, ob Sie Ihren zuvor erstellten Header auf einen weiteren oder alle anderen Positionen kopieren wollen. Kopieren Sie Ihren Header auf alle anderen Positionen, erhalten Sie einen Track, bei dem alle Sektoren gleich sind. Ein so veränderter Track kann nur sehr schwer kopiert werden. Doch darauf werden wir noch zu einem späteren Zeitpunkt eingehen.
5. *Ende, formatieren*
   Formatiert den angegebenen Track. Eine Änderung der Speed-Flags ist möglich.

Das Programm besteht aus drei Teilen. Der eine ist der BASIC-Teil, über den alle Eingaben sowie die Vertauschung der Header vorgenommen werden. Der zweite ist ein Maschinenspracheteil, der die Blockheader, die später auf Disk geschrieben werden, im Computer erstellt. Er entspricht in etwa der Routine, die sich auch in der Formatroutine der Floppy befindet. Der dritte Teil ist ein Maschinenspracheprogramm, das in der Floppy gestartet wird. Es übergibt die Parameter, wie beispielsweise die Länge der Lücke zwischen den Sektoren, an die Formatroutine.

Wie schon gesagt, werden die Blockheader erst im Computer erzeugt, um nach ihrer Änderung in die Floppy ab Adresse $0300 geschickt zu werden, wo sie in das GCR-Format gewandelt und auf Diskette aufgetragen werden. Im folgenden zeigen wir Ihnen die zwei erwähnten Maschinenspracheteile.. Als erstes den Teil, der im Computer abläuft und die Blockheader im Speicher des Rechners erzeugt.

Das Programm, das die Daten für die Blockheader erzeugt, wird ab $C040 (49216) gestartet und legt sie ab $C200 (49664) im Speicher ab.

```
C000 LDA #$08     $08 laden (Zeichen für Blockheader)
C002 STA $C200,Y  und übergeben
C005 INY          Zeiger erhöhen
C006 INY          erhöhen, um Lücke für Prüfsumme zu lassen
C007 LDA $F7      Sektornummer holen
C009 STA $C200,Y  und speichern
C00C INY          Zeiger erhöhen
C00D LDA $F8      Track-Nummer holen
C00F STA $C200,Y  und speichern
C012 INY          Zeiger erhöhen
C013 LDA $FA      zweite ID holen
C015 STA $C200,Y  und übergeben
C018 INY          Zeiger erhöhen
C019 LDA $F9      erste ID holen
C01B STA $C200,Y  und übergeben
C01E INY          Zeiger erhöhen
C01F LDA #$0F     $0F (Lücken-Byte) holen
C021 STA $C200,Y  und speichern
C024 INY          Zeiger erhöhen
C025 STA $C200,Y  und Lücken-Byte erneut speichern
C028 INY          Zeiger erhöhen
C029 LDA #$00     Akku mit $00 laden (normalisieren)
C02B EOR $C1FA,Y
C02E EOR $C1FB,Y  Prüfsumme über den
C031 EOR $C1FC,Y  Blockheader berechnen
C034 EOR $C1FD,Y
C037 STA $C1F9,Y  und Prüfsumme abspeichern
C03A RTS          Rücksprung
```

Für die Erstellung der Blockheader wird das Programm hier gestartet.

```
C040 LDA #$00     Sektornummer auf 0 stellen
C042 STA $F7
C044 LDY #$00     Zeiger auf 0 setzen
C046 JSR $C000    und Blockheader erstellen
C049 INC $F7      Sektornummer erhöhen
C04B LDA $F7      Nummer laden
C04D CMP $FB      und mit maximaler Zahl vergleichen.
C04F BCC $C046    verzweige, wenn Maximum nicht erreicht
C051 RTS          Rücksprung
```

Für die Errechnung der Prüfsumme eines einzelnen Blockheaders wird die Routine hier gestartet.

```
C052 LDA #$00     Position des Headers laden (variabel)
C054 ASL
C055 ASL          Mit 8 multiplizieren
C056 ASL
C057 CLC          Carry für Addition löschen
```

```
C058 ADC #$08     und mit 8 addieren,
C05A TAY          als Zählwert übergeben
C05B JMP $C029    Prüfsumme errechnen
```

Die folgende Routine wird in der Floppy ab $0400 gestartet, nachdem die Blockheader ebenfalls in die Floppy gesandt wurden. Alle nötigen Parameter werden mit der zu startenden Routine mitgesandt. Sie wird ab $0424 gestartet.

```
0400 LDY #$88     Low-Byte der Endadresse der Blockheader
0402 LDX #$00     X-Register löschen
0404 LDA #$23     Zeiger auf Ende der Formatierung setzen.
0406 STA $51      damit nur ein Track formatiert wird
0408 LDA #$11     Maximale Sektoranzahl laden (variabel)
040A STA $43      und übergeben
040C STA $0628
040F LDA #$0B     Lücke zwischen den Sektoren laden
0411 STA $0626    und übergeben
0414 STA $0620    gleichzeitig als Zähler für Fehler nehmen
0417 LDA $1C00    Control-Port laden
041A AND #$9F     Speed-Flags löschen
041C ORA #$00     Mit eigenem Speed verknüpfen (hier 00)
041E STA $1C00    und speichern
0421 JMP $FC84    Formatierung beginnen
```

Das Programm wird hier gestartet.

```
0424 LDA #$23     Nummer des zu formatierenden Tracks
0426 STA $08      übergeben
0428 LDA #$E0     Job-Code E0 für Programm starten
042A STA $01      in Job-Speicher schreiben
042C LDA $01      Code für Rückmeldung empfangen?
042E BMI $042C    verzweige, wenn nicht empfangen
0430 RTS          Rücksprung
```

Nachdem wir nun das Programm erklärt haben, wollen wir uns nun mit seiner Anwendung beschäftigen.

5.8.5 Das Arbeiten mit zerstörten Blöcken

Wie wir schon sagten, kann man mit dem obigen Programm die Blockheader-Parameter ändern und dadurch Sektoren gezielt zerstören. Dieses Verfahren dient, bis auf einige Sonderfälle, kaum zum Schutz gegen das Kopieren, sondern nur zur Verhinderung des Zugriffs anderer auf einen so geschützten Block.

Jetzt stellt sich jedoch die Frage, wie man überhaupt Daten auf diesen Blöcken speichern oder wieder lesen soll, denn die Floppy fängt jedesmal an zu blinken, wenn man versucht, den Block auf die herkömmliche Weise zu laden.

Sie können sich sicher denken, daß es natürlich einen Weg gibt, diese Blöcke zu nutzen. Die einfachste Methode, die sich auch für alle Arten von zerstörten Blockheadern einsetzen läßt, ist die: Man liest den Blockheader vor dem zerstörten Header ein. Die darauffolgende SYNC-Markierung ist folglich die des Datenblocks des gelesenen Headers. Wir überspringen mit unserem Floppy-Programm einfach diese Markierung. Die nächste SYNC-Markierung, die gefunden wird, ist dann die unseres zerstörten Blockheaders. Diese Markierung wird ebenfalls nicht beachtet. Die daraufhin gefundene SYNC-Markierung muß dann die des Datenblocks sein, der zu dem zerstörten Header gehört. Nachdem diese Markierung gefunden wurde, wird der Datenblock gelesen. Sie sehen also, daß es uns mit diesem Trick möglich ist, Daten von einem auch noch so zerstörten Block zu lesen, was uns sonst nicht möglich gewesen wäre. Nach dem gleichen System können wir auch Daten speichern. Sollten Sie mehrere Blockheader hintereinander zerstört haben, so brauchen Sie nur entsprechend viele SYNC-Markierungen zu überlesen.

Mit dem folgenden Floppy-Programm ist es möglich, eine beliebige Anzahl von Sektoren zu überspringen und dann den gewünschten Block zu lesen. Das Programm wird ab $0522 gestartet.

```
0500 JSR $F50A   Blockheader suchen, auf Datenblock warten
0503 LDX #$01    Zähler für Sektor überspringen (variabel)
0505 JSR $0512   Routine zum Überspringen eines Sektors
0508 DEX         Zähler verringern
0509 BNE $0505   verzweige, wenn nicht abgelaufen
050B LDA #$06    High-Byte für Pufferzeiger auf $06 stellen
050D STA $31     damit die Daten ab $0600 geladen werden
050F JMP $F4D4   in Routine für Block laden springen

0512 LDA $1C00   Control-Port laden
0515 BPL $0512   verzweige, wenn SYNC noch anliegt
0517 JSR $F556   auf neue SYNC-Markierung warten
051A LDA $1C00   Control-Port laden
```

```
051D BPL $051A   verzweige, wenn SYNC noch anliegt
051F JMP $F556   auf neue SYNC-Markierung warten, RTS
```

Das Programm wird hier gestartet.

```
0522 LDA #$01    Track auf dem geladen wird (variabel)
0524 STA $0A     übergeben
0526 LDA #$00    Sektor, der geladen wird (variabel)
0528 STA $0B     übergeben
052A LDA #$E0    Job-Code für Programm ausführen, laden
052C STA $02     und übergeben
052E LDA $02     Rückmeldung abwarten
0530 BMI $052E   verzweige, wenn keine Rückmeldung
0532 RTS         Rücksprung
```

Im Anschluß daran das gewohnte BASIC-Listing. Es erlaubt die Eingabe des zu lesenden Tracks, des Sektor Sektors sowie die Eingabe der zu überspringenden Sektoren. Die Nummer des tatsächlich gelesenen Sektors, ergibt sich aus der Nummer des eingegebenen Sektors, addiert mit der Anzahl der zu überspringenden Sektoren. Die Daten des zu lesenden Blocks werden ab $0600 (Puffer 3) in der Floppy abgelegt.

```
10 OPEN1,8,15,"I"
20 READ X:IF X=-1THEN 100
25 SU=SU+X
30 PRINT#1,"M-W"CHR$(N)CHR$(5)CHR$(1)CHR$(X)
40 N=N+1:GOTO 20
100 IFSU <> 5477 THEN PRINT"ERROR IN DATAS":STOP
105 INPUT "WELCHER TRACK";T
110 INPUT "WELCHER SEKTOR";S
115 INPUT "ZU UEBERLESENE SEKTOREN";U
120 PRINT#1,"M-W"CHR$(35)CHR$(5)CHR$(1)CHR$(T)
125 PRINT#1,"M-W"CHR$(39)CHR$(5)CHR$(1)CHR$(S)
130 PRINT#1,"M-W"CHR$(4)CHR$(5)CHR$(1)CHR$(U)
135 PRINT#1,"M-E"CHR$(34)CHR$(5)
140 FORN=1TO 500:NEXT
145 PRINT#1,"M-R"CHR$(2)CHR$(0)CHR$(1)
150 GET#1,A$:A=ASC (A$+CHR$(0))
160 IF A>127 THEN130
170 IFA =1 THENPRINT"OK":END
180 PRINT"ERROR":END
302 DATA 32, 10,245,162, 1, 32, 18, 5,202,208,250,169, 6,133, 49, 76,212
,244
304 DATA173, 0, 28, 16,251, 32, 86,245,173, 0, 28, 16,251, 76, 86,245,16
9, 1
306 DATA133, 10,169, 0,133, 11,169,224,133, 2,165, 2, 48,252, 96, -1
```

Wie das Lesen eines Blocks, so erfolgt auch das Schreiben. Das hierzu benötigte Programm sieht jedoch etwas anders aus, da die SAVE-Routine der Floppy später angesprungen werden muß. Die von ihr abgearbeiteten Programmteile müssen zum Teil in das eigene Programm übernommen werden.

Hier nun die entsprechende SAVE-Routine, die ab $0400 in der Floppy gestartet wird.

```
0500 LDA #$06     High-Byte des Puffers auf $06 stellen,
0502 STA $31      um die Daten von $0600 zu speichern
0504 JSR $F5E9    Prüfsumme über Datenblock berechnen
0507 STA $3A      und abspeichern
0509 LDA $1C00    Control-Port laden
050C AND #$10     und auf Schreibschutz prüfen
050E BNE $0515    verzweige, wenn kein Schreibschutz
0510 LDA #$08     Fehlermeldung im Akku
0512 JMP $F969    und zur Ausgabe springen
0515 JSR $F78F    Pufferinhalt ins GCR-Format wandeln
0518 JSR $F510    Blockheader suchen
051B LDX #$01     Zähler für Sektor überspringen (variabel)
051D JSR $0526    Routine zum Überspringen eines Sektors
0520 DEX          Zähler verringern
0521 BNE $051D    verzweige, wenn nicht abgelaufen
0523 JMP $F58C    Block auf Diskette schreiben

0526 LDA $1C00    Control-Port laden
0529 BPL $0526    verzweige, wenn SYNC-Signal noch anliegt
052B JSR $F556    auf nächste SYNC-Markierung warten
052E LDA $1C00    Control-Port lesen
0531 BPL $052E    verzweige, wenn SYNC-Signal noch anliegt
0533 JMP $F556    auf nächste SYNC-Markierung warten, RTS
```

Das Programm wird hier gestartet.

```
0536 LDA #$01     Track, auf dem geladen wird (variabel),
0538 STA $0A      übergeben
053A LDA #$00     Sektor, der geladen wird (variabel),
053C STA $0B      übergeben
053E LDA #$E0     Job-Code für Programm ausführen, laden
0540 STA $02      und übergeben
0542 LDA $02      Rückmeldung abwarten
0544 BMI $0542    verzweige, wenn keine Rückmeldung
0546 RTS          Rücksprung
```

Jetzt folgt wieder der BASIC-Loader, bei dem Sie Track und Sektor sowie die Anzahl der zu überspringenden Blöcke eingeben können.

```
10 OPEN1,8,15,"I"
20 READ X:IF X=-1THEN 100
25 SU=SU+X
30 PRINT#1,"M-W"CHR$(N)CHR$(5)CHR$(1)CHR$(X)
40 N=N+1:GOTO 20
100 IFSU <> 7633 THEN PRINT"ERROR IN DATAS":STOP
105 INPUT "WELCHER TRACK";T
110 INPUT "WELCHER SEKTOR";S
115 INPUT "ZU UEBERLESENE SEKTOREN";U
120 PRINT#1,"M-W"CHR$(55)CHR$(5)CHR$(1)CHR$(T)
125 PRINT#1,"M-W"CHR$(59)CHR$(5)CHR$(1)CHR$(S)
130 PRINT#1,"M-W"CHR$(28)CHR$(5)CHR$(1)CHR$(U)
135 PRINT#1,"M-E"CHR$(54)CHR$(5)
140 FORN=1TO 500:NEXT
145 PRINT#1,"M-R"CHR$(2)CHR$(0)CHR$(1)
150 GET#1,A$:A=ASC (A$+CHR$(0))
160 IF A>127 THEN130
170 IFA =1 THENPRINT"OK":END
180 PRINT"ERROR":END
302 DATA169,  6,133, 49, 32,233,245,133, 58,173,  0, 28, 41, 16,208,  5,
169,  8
304 DATA 76,105,249, 32,143,247, 32, 16,245,162, 1, 32, 38,  5,202,208,2
50, 76
306 DATA140,245,173, 0, 28, 16,251, 32, 86,245,173, 0, 28, 16,251, 76, 8
6,245
308 DATA169, 1,133, 10,169, 0,133, 11,169,224,133, 2,165, 2, 48,252, 96,
 -1
```

Nachdem wir gesehen haben, wie wir uns mit Hilfe von Blockheader-Manipulationen vor dem Zugriff Fremder auf unsere Blöcke schützen können, wollen wir uns einmal ansehen, wie man mit dieser Methode einen sehr guten Kopierschutz aufbauen kann. Bei der Änderung von mehreren Blockheadern auf dem gleichen Track kann es zu Störungen beim Einlesen eines Blocks kommen.

5.8.6 Gleiche Blöcke auf einem Track

Eine sehr sichere Kopierschutzmethode ist es, einen Track nur mit gleichen Blockheadern zu beschreiben. Einen solchen Track kann man mit dem schon beschriebenen Formatierungsprogramm mühelos erstellen.

Wie arbeitet nun ein solcher Kopierschutz und warum ist er so überaus schwer, fast unmöglich, zu kopieren? Um dies zu ver-

stehen, müssen wir uns aber noch etwas näher mit dem System der Kopierprogramme beschäftigen.

Das Kopierprogramm lädt einen oder mehrere Blöcke zugleich in den Speicher der Floppy und schickt daraufhin die Daten zum Computer, da der Speicher der Floppy nicht ausreicht, weitere Daten zu speichern. Durch das Senden der Daten an den Computer muß das Kopierprogramm die Stelle wiederfinden, an der es aufgehört hat, Daten zu lesen. Es sucht wieder die Anfangsstelle und überliest eine entsprechende Anzahl von Daten, um daraufhin die neuen Daten einzulesen. Das Programm erkennt einen schon vollständig eingelesenen Track daran, daß es dieselben Daten findet, die schon zu Beginn eingelesen wurden. Wenn Sie sich jetzt daran erinnern, wie unser modifizierter Track aussieht, werden Sie feststellen, daß das Kopierpogramm keine Möglichkeit hat, sich auf dem Track zu orientieren, da alle Blöcke gleich aussehen. Für das Programm ist es somit fast unmöglich, die Anzahl der SYNC-Markierungen sowie die Anzahl der Bytes auf diesem Track festzustellen. Auch die Kopierprogramme, die mit einem parallelen Bus arbeiten, welcher die Daten ohne "abzusetzen" einlesen kann, bekommen Probleme, die Anzahl der SYNC-Markierungen richtig festzustellen.

Dieser Track ist so wirkungsvoll, weil es für Kopierprogramme sehr schwer ist, ihn richtig zu analysieren. Wir wissen also, womit die Kopierprogramme zu "kämpfen" haben. Doch stellt sich hier die Frage, wie wir selbst einen solchen Track auf seine Richtigkeit überprüfen.

Diese Überprüfung ist recht aufwendig, da sie in drei Etappen durchgeführt wird. Als erstes wird eine große Anzahl von Bytes eingelesen und gleichzeitig die Anzahl der SYNC-Markierungen gezählt. Diese Zahl ist, falls der Track korrekt ist, recht konstant. Als nächstes wird überprüft, wie lang die SYNC-Markierungen selbst sind, damit das Kopierprogramm nicht, wenn es nicht genügend Daten feststellt, die entstehende Lücke mit längeren SYNC-Markierungen vertuschen kann. Als drittes und letztes muß noch geprüft werden, ob die Blockheader des entsprechenden Tracks auch wirklich alle gleich sind. Sollten alle Prüfungen positiv ausfallen, so liegt ein "Original" vor, was

durch eine entsprechende Rückmeldung von der Kopierschutzabfrage signalisiert wird. Ein Programm, das all diese Prüfdurchgänge enthält, werden Sie jetzt anschließend sehen. Es liegt (in der Floppy) ab Adresse $0400 im Speicher und wird ab $04DF gestartet. Die entsprechende Rückmeldung an den Rechner wird in $10 der Floppy gespeichert. 0 steht für "Kopierschutz nicht richtig erkannt" und $FF für "alles OK".

```
0400 LDA $08       Nummer des zu prüfenden Tracks laden
0402 LDX #$04      Zähler für die vier Zonenabschnitte
0404 CMP $04DA,X   Nummer mit Zahl aus Tabelle vergleichen
0407 DEX           Zähler verringern
0408 BCS $0404     verzweige, wenn Nummer größer/gleich der
                   Zahl aus der Tabelle
040A TXA           entsprechenden Zählerwert in den Akku
040B ASL           schieben und dann die Bits an Position
040C ASL           5 und 6 schieben, um sie als Maßzahl
040D ASL           für den Speed auf dem entsprechenden
040E ASL           Track zu benutzen
040F ASL
0410 STA $44       Wert zwischenspeichern
0412 LDA $1C00     Control-Port laden
0415 AND #$9F      Bits für Speed löschen
0417 ORA $44       mit eigenem Speed verknüpfen
0419 STA $1C00     und Wert übergeben
041C LDX #$00      Low- und High-Byte der Anzahl der zu
041E LDY #$1F      lesenden Bytes laden (7936)
0420 LDA #$00      Zählwert für SYNC auf
0422 STA $37       Null setzen
0424 LDA $1C00     Control-Port laden
0427 BPL $0424     verzweige, wenn SYNC noch anliegt
0429 CLV           Byte-Ready-Leitung löschen
042A BVC $042A     warten, bis Byte anliegt
042C CLV           Byte-Ready-Leitung wieder löschen
042D DEX           Byte-Zähler verringern
042E BNE $0433     verzweige, wenn kein Übertrag
0430 DEY           sonst auch High-Byte verringern
0431 BEQ $043D     verzweige, wenn Zähler abgelaufen
0433 LDA $1C00     Control-Port laden
0436 BMI $0429     verzweige, wenn kein SYNC anliegt
0438 INC $37       sonst SYNC-Zähler erhöhen
043A JMP $0424     und auf neue SYNC-Markierung warten
043D LDX $37       Zähler für gefundene SYNC-Markierungen
043F STX $04FF     in $04FF speichern
```

Der jetzt folgende Programmteil prüft die Länge der SYNC-Markierungen. Zu diesem Zweck wird der Timer 1 der Ein/Ausgabe VIA benutzt.

```
0442 LDA #$C0     Low-Byte des Zählwerts für SYNC-Länge
0444 STA $1804    ins Low-Byte des Timers schreiben
0447 LDA $1C00    Control-Port laden und auf Ende
044A BPL $0447    der SYNC-Markierung warten
044C LDA $1C00    Control-Port laden und auf Anfang
044F BMI $044C    der SYNC-Markierung warten
0451 LDA #$80     High-Byte des Timerwertes laden und
0453 STA $1805    übergeben, Timer starten
0456 LDA $1C00    Control-Port laden und auf Ende
0459 BPL $0456    der SYNC-Markierung warten
045B LDA $1805    High-Byte des Timerwertes laden
045E BPL $0465    verzweige, wenn Wert kleiner $80 (falsch)
0460 DEX          Zähler für Anzahl der SYNC verringern
0461 BNE $0442    und nächste Markierung abfragen
0463 LDA #$FF     Rückmeldewert laden
0465 STA $04FE    und in $04FE speichern
```

Als nächstes folgt der Teil, in dem 25 aufeinanderfolgende Blockheader geladen und ab $0500 gespeichert werden. Es gibt zwar im Normalfall keine 25 verschiedenen Blockheader auf einem Track, jedoch geht man so sicher, daß alle geladen werden. Dieses Teilprogramm kann auch separat verwendet werden, um die Reihenfolge der Sektoren oder ähnliches zu überprüfen.

```
0468 LDX #$00     Zähler auf Null setzen
046A JSR $F556    Auf SYNC-Markierung warten
046D LDY #$09     Zähler für Anzahl der Bytes pro Header
046F BVC $046F    warten, bis ein Byte anliegt
0471 CLV          Byte-Ready-Leitung löschen
0472 LDA $1C01    Byte vom Port holen
0475 CMP #$52     mit Wert für Blockheader vergleichen
0477 BNE $046A    verzweige, wenn kein Blockheader
0479 STA $0500,X  sonst Wert speichern
047C INX          Zähler erhöhen
047D NOP
047E BVC $047E    auf neues Byte warten
0480 CLV          Byte-Ready-Leitung löschen
0481 LDA $1C01    Byte vom Port holen
0484 STA $0500,X  und speichern
0487 INX          Zähler erhöhen
0488 BEQ $048F    verzweige, wenn alle Header geholt
048A DEY          Zähler für Bytes pro Header verringern
048B BNE $047E    verzweige, wenn nicht alle Bytes geholt
048D BEQ $046A    unbedingter Sprung
048F LDA #$05     High-Byte des zu bearbeitenden Puffers auf
0491 STA $31      $05 stellen (Puffer 2)
0493 JSR $F8E0    Bytes ins Bin.-Format wandeln
0496 LDX #$FF     Zähler setzen, um die Daten umzukopieren
0498 LDA $04FF,X  Daten holen
049B STA $0500,X  und ein Byte weiter abspeichern
```

```
049E DEX           Zähler verringern
049F CPX #$FF      vergleiche, ob schon alles kopiert
04A1 BNE $0498     verzweige, wenn nicht alles kopiert
04A3 LDA $38       erstes Byte, das bei der Umwandlung ins
04A5 STA $0500     Bin.-Format gespeichert wurde, zurück-
                   schreiben
```

Dieser Teil dient zur Überprüfung der gelesenen Blockheader.

```
04A8 LDY #$C8      Zeichenanzahl laden
04AA LDX #$08      Zeichenanzahl pro Header
04AC LDA $04FF,X Zeichen holen
04AF CMP $04FF,Y mit erstem Header vergleichen
04B2 BNE $04C6     verzweige, wenn Zeichen ungleich
04B4 DEX           Zähler verringern
04B5 DEY           Zähler verringern
04B6 BEQ $04BE     verzweige, wenn alle Zeichen verglichen
04B8 CPX #$00      kontrollieren, ob ganzer Header verglichen
04BA BNE $04AC     verzweige, wenn nein
04BC BEQ $04AA     unbedingter Sprung
04BE LDA $04FE     Rückmeldung der SYNC-Prüfroutine holen
04C1 CMP #$FF      auf Richtigkeit überprüfen
04C3 BNE $04D7     verzweige, wenn Fehler
04C5 LDA $04FF     Anzahl der gefundenen Sektoren holen
04C8 CMP #$2A      mit 42 vergleichen
04CA BCC $04D7     Fehler, wenn der Wert kleiner ist
04CC CMP #$2E      Wert mit 46 vergleichen
04CE BCS $04D7     Fehler, wenn Wert größer oder gleich ist
04D0 LDA #$FF      Positive Rückmeldung an Computer übergeben
04D2 STA $10       Rückmeldung schreiben
04D4 JMP $FD9E     Rücksprung
04D7 LDA #$00      negative Rückmeldung laden
04D9 BEQ $04D2     unbedingter Sprung
.
04DB 2B 1F 19 12 Werte für die vier Zonenabschnitte
```

Das Programm wird hier gestartet.

```
04DF LDA #$01      Track-Nummer laden
04E1 STA $08       und an Job übergeben
04E3 LDA #$E0      Job-Code E0 laden (Programm ausführen)
04E5 STA $01       in Job-Seicher schreiben
04E7 LDA $01       Rückmeldung abwarten
04E9 BMI $04E7     verzweige, wenn noch keine Rückmeldung
04EB RTS           Rücksprung
```

In diese Routine wurde eine eigene Routine eingebaut, die den Speed des jeweiligen Tracks bestimmt, da diese Routine im Betriebssystem der Floppy diese Aufgabe nur bis Track 35 fehler-

los durchführt. Im Anschluß an das Maschinenprogramm folgt wieder unser BASIC-Loader.

```
10 OPEN1,8,15,"I"
20 READ X:IF X=-1THEN 100
25 SU=SU+X
30 PRINT#1,"M-W"CHR$(N)CHR$(4)CHR$(1)CHR$(X)
40 N=N+1:GOTO 20
100 IFSU <> 28329 THEN PRINT"ERROR IN DATAS":STOP
110 INPUT "WELCHER TRACK";T
120 PRINT#1,"M-W"CHR$(224)CHR$(4)CHR$(1)CHR$(T)
130 PRINT#1,"M-E"CHR$(223)CHR$(4)
135 FORN=1TO 500:NEXT
140 PRINT#1,"M-R"CHR$(16)CHR$(0)CHR$(1)
150 GET#1,A$:A=ASC (A$+CHR$(0))
170 IFA =255 THENPRINT"SCHUTZ OK":END
180 PRINT"SCHUTZ ERROR":END
302 DATA165, 8,162, 4,221,218, 4,202,176,250,138, 10, 10, 10, 10, 10,133
, 68
304 DATA173, 0, 28, 41,159, 5, 68,141, 0, 28,162, 0,160, 31,169, 0,133,
55
306 DATA173, 0, 28, 16,251,184, 80,254,184,202,208, 3,136,240, 10,173,
0, 28
308 DATA 48,241,230, 55, 76, 36, 4,166, 55,142,255, 4,169,192,141, 4, 24
,173
310 DATA 0, 28, 16,251,173, 0, 28, 48,251,169,128,141, 5, 24,173, 0, 28,
 16
312 DATA251,173, 5, 24, 16, 5,202,208,223,169,255,141, 254, 4,162, 0, 32
, 86
314 DATA245,160, 9, 80,254,184,173, 1, 28,201, 82,208,241,157, 0, 5,232,
234
316 DATA 80,254,184,173, 1, 28,157, 0, 5,232,240, 5,136,208,241,240,219,
169
318 DATA  5,133, 49, 32,224,248,162,255,189,255,  4,157,  0,  5,202,224,
255,208
320 DATA245,165, 56,141, 0, 5,160,200,162, 8,189,255, 4,217,255, 4,208,
18
322 DATA202,136,240, 6,224, 0,208,240,240,236,173,254, 4,201,255,208, 18
,173
324 DATA255, 4,201, 42,144, 11,201, 46,176, 7,169,255,133, 16, 76,158,25
3,169
326 DATA  0,240,247, 43, 31, 25, 18,169, 1,133, 8,169,224,133, 1,165, 1,
 48
328 DATA252, 96, -1
```

Aufbringen können Sie diesen Schutz, indem Sie das Programm zum Ändern der Blockheader-Parameter, welches sich weiter vorne befindet, verwenden und dort, wie in der Anleitung schon erwähnt, "Kopiere Header" anwählen, um daraufhin den entsprechenden Header auf *alle* anderen Positionen zu kopieren.

6. Die Wunderwelt der Grafik

Die Grafikfähigkeiten des Commodore 64 lassen sich grob in vier Bereiche unterteilen:

Textbildschirm mit Blockgrafik

Auf dem normalen Textbildschirm, mit dem Sie ja für gewöhnlich arbeiten, steht Ihnen eine Vielzahl von Grafikzeichen zur Verfügung, die wie die "normalen" Zeichen (Buchstaben, Ziffern usw.) auf den Bildschirm gebracht werden. Sicher haben Sie mit diesen Grafikzeichen in der Zwischenzeit schon fleißig experimentiert. Die Abbildungen der einzelnen Zeichen auf der Tastatur laden ja geradezu dazu ein.

Mit ein wenig Geschick lassen sich damit schon recht ansprechende Grafiken erzeugen. Da die Auflösung der Grafiken mit 40 (Zeilen) mal 25 (Spalten) aber nur sehr grob ist, spricht man hier auch von einer sogenannten "Blockgrafik".

Grafikbildschirm mit hochauflösender Grafik

Wie der Name schon vermuten läßt, bietet die "hochauflösende Grafik" eine wesentlich höhere Auflösung als die Blockgrafik. Je nachdem, mit wieviel Farben man arbeiten möchte, steht einem eine Auflösung von 320*200 Punkten (2 Farben) oder 160*200 Punkten (4 Farben) zur Verfügung. Alle kommerziellen Grafikprogramme arbeiten in einer dieser beiden Auflösungen.

Sprites

Sprites, unentbehrliches "Hilfsmittel" bei fast allen Spielen, sind kleine frei bewegliche Grafikobjekte, die unabhängig vom Textbildschirm oder der hochauflösenden Grafik verwaltet werden. Der Commodore 64 ist in der Lage, bis zu acht Sprites gleichzeitig auf dem Bildschirm darzustellen.

Selbstdefinierte Zeichen

Das Aussehen sämtlicher auf dem Textbildschirm darstellbarer Zeichen (Buchstaben, Ziffern, Sonderzeichen und Grafikzeichen) ist in einem speziellen Speicherbereich des Commodore 64, dem sogenannten "Zeichengenerator" oder "Zeichensatz-ROM", abgelegt. Der Zeichengenerator läßt sich leicht auslesen und in einem anderen Speicherbereich unterbringen. Dadurch kann man sich ohne weiteres eigene Zeichen definieren oder vorhandene Zeichen abändern, etwa um die deutschen Umlaute ä, ö und ü zur Verfügung zu haben.

Für sämtliche Grafikfähigkeiten des Commodore 64 ist ein nur einzelner Baustein innerhalb des 64'ers verantwortlich, der sogenannte "VIC-II-Chip".

Leider unterstützt das BASIC 2.0 die drei letztgenannten Bereiche nur sehr unzureichend, um nicht zu sagen, überhaupt nicht. Die einzigen Hilfsmittel, die einem zur Verfügung stehen sind POKE und PEEK.

Wenn Sie schon einmal einen Blick in das Grafikkapitel des Handbuchs geworfen haben, werden Sie festgestellt haben, daß beispielsweise allein schon das bloße Einschalten der hochauflösenden Grafik oder das Setzen eines Punktes sehr lange POKE- und PEEK-Sequenzen erfordert. Damit läßt sich natürlich nicht vernünftig arbeiten.

Ich werde Ihnen deshalb im Laufe dieses Kapitels eine Vielzahl von Maschinensprache-Routinen vorstellen, die Sie bequem von BASIC aus aufrufen können. Alle theoretischen Erklärungen zur Erzeugung der Grafik durch den VIC-II-Chip beschränken sich daher auf das wirklich notwendige, das Sie zur sinnvollen Anwendung der Routinen benötigen.

6.1 Einfache Blockgrafik

Beginnen wir mit dem einfachsten, der Blockgrafik. Wie Sie vielleicht wissen, ist das Bild eines Fernsehers oder Monitors

nicht von Dauer. Es muß bis zu 50 mal in der Sekunde neu aufgebaut werden. Der Commodore 64 muß den Fernseher oder Monitor also ständig mit Informationen darüber versorgen, was dieser auf dem Bildschirm darstellen soll. Um diese Informationen aber überhaupt bereitstellen zu können, muß der 64'er "wissen", welche Bildschirmpositionen mit welchen Zeichen und welcher Farbe belegt sind.

Dazu gibt es im Commodore 64 zwei spezielle Speicherbereiche: den "Bildschirmspeicher" (für die Zeichen) und den "Farbspeicher". Der Bildschirmspeicher erstreckt sich von Speicheradresse 1024 bis 2023, der Farbspeicher von Adresse 55296 bis 56295. Beide Bereiche sind also genau 1000 (40*25) Byte groß.

Wenn Sie mit PRINT Zeichen auf den Bildschirm bringen, müssen Sie sich um diese beiden Speicherbereiche nicht kümmern. Und für einfache Bildschirmmasken, etwa für eine Dateiverwaltung, reicht PRINT in der Regel auch vollkommen aus. Ein großer Nachteil von PRINT ist aber die fehlende Möglichkeit, gezielt eine bestimmte Bildschirmposition ansteuern zu können. Außerdem kommt einem beim Aufbau einer Bildschirmmaske häufig ein unerwünschtes Scrollen des Bildschirms um eine Zeile nach unten (wenn man ganz am Ende einer Bildschirmzeile etwas schreibt) oder eine Zeile nach oben (wenn man sich in der letzten Bildschirmzeile befindet) in die Quere.

Um das zu vermeiden, bedient man sich am besten des Befehls POKE. Mit POKE kann man direkt in jede Speicherzelle des Bildschirm- und Farbspeichers hineinschreiben und dadurch am Bildschirm Zeichen und Farben setzen. Leider unterscheiden sich die Zeichencodes im Bildschirmspeicher aber von dem sonst gebräuchlichen ASCII-Code. Daher kann man beispielsweise nicht einfach schreiben

```
POKE ....,ASC("A")
```

sondern muß jedesmal erst nachschauen, welchen Bildschirmcode das betreffende Zeichen hat. Eine vollständige Tabelle der Bildschirmcodes finden Sie dazu zusammen mit dem ASCII-Code im Anhang. Mit Hilfe von POKE in den Farbspeicher kann man

sehr schnell und auch bequem für etwas mehr Farbenpracht auf dem sonst eintönigen Bildschirm sorgen.

Ein Nachteil bei beiden Speicherbereichen sind aber die relativ unhandlichen Speicheradressen. Die folgende Tabelle gibt Ihnen einen Überblick, ab welchen Speicheradressen die Bildschirmzeilen beginnen:

Bildschirmzeile	Bildschirmspeicher	Farbspeicher
0	1024	55296
1	1064	55336
2	1104	55376
3	1144	55416
4	1184	55456
5	1224	55496
6	1264	55536
7	1304	55576
8	1344	55616
9	1384	55656
10	1424	55696
11	1464	55736
12	1504	55776
13	1544	55816
14	1584	55856
15	1624	55896
16	1664	55936
17	1704	55976
18	1744	56016
19	1784	56056
20	1824	56096
21	1864	56136
22	1904	56176
23	1944	56216
24	1984	56256

Die Adresse zu einer bestimmten Zeilen-/Spaltenposition läßt sich auch leicht berechnen:

```
BD=1024+40*ZE+SP
```

für den Bildschirmspeicher und

```
FD=55296+40*ZE+SP
```

für den Farbspeicher, wobei ZE die Zeile (0-24) und SP die Spalte (0-39) enthält. Um Ihnen das ganze einmal praktisch zu demonstrieren, zeichnet das folgende Programm einen Rahmen um den Bildschirm. Als Zeichen habe ich den Stern [*] (Bildschirmcode 42) gewählt. Die Farbe des Rahmens wird für jede Position zufällig gesetzt:

```
100 rem rahmen zeichnen
105 zc=42:rem '*'
110 :
120 rem oberer teil
120 for sp=0 to 39
130 poke 1024+sp,zc
140 poke 55296+sp,rnd(0)*16
150 next sp
160 :
120 rem rechter teil
120 for ze=0 to 24
130 poke 1024+40*ze+39,zc
140 poke 55296+40*ze+39,rnd(0)*16
150 next ze
160 :
120 rem unterer teil
120 for sp=39 to 0 step -1
130 poke 1984+sp,zc
140 poke 56256+sp,rnd(0)*16
150 next sp
160 :
120 rem linker teil
120 for ze=24 to 0 step -1
130 poke 1024+40*ze,zc
140 poke 55296+40*ze,rnd(0)*16
150 next ze
160 :
```

Starten Sie das Programm einmal. Das sieht doch schon recht bunt aus. Zur Farbgebung stehen Ihnen, wie Sie ja wissen, insgesamt 16 Farben zur Verfügung:

```
0: schwarz   8: orange
1: weiß      9: braun
2: rot      10: hellrot
3: türkis   11: grau 1
4: violett  12: grau 2
5: grün     13: hellgrün
6: blau     14: hellblau
7: gelb     15: grau 3
```

Im Normalfall kann man für jede Bildschirmposition (und damit für jedes dargestellte Zeichen) genau eine Farbe definieren. Die Hintergrundfarbe wird mit

```
POKE 53281,FC
```

für den gesamten Bildschirm festgelegt. Es gibt aber zwei spezielle Darstellungsmodi, in denen man den Bildschirm bunter gestalten kann, der sogenannte "Erweiterte-Hintergrund-Farbmodus" und der "Multicolor-Modus".

Im Erweiterten-Hintergrund-Farbmodus stehen jeweils vier Farben (anstatt einer) zur Verfügung. Die Farbcodes dieser Farben müssen in den Speicherzellen 53281 bis 53284 abgelegt werden. Welche Hintergrundfarbe an den einzelnen Bildschirmpositionen erscheint, hängt vom Bildschirmcode der am Bildschirm befindlichen Zeichen ab. Entscheidend sind dabei die letzten beiden Bits des Codes. Die folgende Tabelle verdeutlicht den Zusammenhang:

Bit 6	Bit 7	Farb-Register
0	0	53281
0	1	53282
1	0	53283
1	1	53284

Um in den Erweiterten-Hintergrund-Farbmodus zu gelangen, geben Sie ein:

```
POKE 53265,PEEK(53265) OR 64
```

Mit

```
POKE 53265,PEEK(53265) AND 191
```

schalten Sie wieder auf die Normaldarstellung um. Da die Bits 6 und 7 nun als Farbzeiger dienen, stehen für den eigentlichen Zeichencode nur noch die Bits 0 bis 5 zur Verfügung. Mit 6 Bits

lassen sich jedoch nur 64 Zeichen codieren. Im Erweiterten-Hintergrund-Farbmodus können daher nur 64 Zeichen dargestellt werden:

Code-Bits 7 6 5 4 3 2 1 0	Zeichen
0 0 0 0 0 0	@
0 0 0 0 0 1	A
0 0 0 0 1 0	B
0 0 0 0 1 1	C
....	...
1 1 1 1 0 1	=
1 1 1 1 1 0	>
1 1 1 1 1 1	?

Nehmen wir zum Beispiel den Buchstaben "A". Er hat den Bildschirmcode 1, binär "00000001". Schreibt man diesen Wert in die Speicherzelle 1024, wird das "A" mit der aktuellen Hintergrundfarbe dargestellt. Ändert man den Wert nun in 193 (binär 11000001) um, wird die Hintergrundfarbe für das "A" aus der Speicherzelle 53284 geholt. Das folgende Programm schreibt alle darstellbaren Zeichen mit den möglichen Hintergrundfarben auf den Bildschirm:

```
100 rem erweiterter-hintergrund-farbmodus ein
110 poke 53265,peek(53265) or 64
120 :
130 poke 53281,0:rem hintergrundfarbe 1
140 poke 53282,1:rem hintergrundfarbe 2
150 poke 53283,2:rem hintergrundfarbe 3
160 poke 53284,3:rem hintergrundfarbe 4
170 for z=0 to 63
180 poke 1024+z,z:rem zeichen mit hintergrundfarbe 1
190 poke 1104+z,z+64:rem zeichen mit hintergrundfarbe 2
200 poke 1184+z,z+128:rem zeichen mit hintergrundfarbe 3
210 poke 1264+z,z+192:rem zeichen mit hintergrundfarbe 4
220 next z
230 :
240 get eg$:if eg$="" then 240:rem auf tastendruck warten
250 :
260 rem erweiterter-hintergrund-farbmodus aus
270 poke 53265,peek(53265) and 191
```

Im Multicolor-Modus nun wird die Zeichenfarbe "vervierfacht", d.h., ein einzelnes Zeichen kann aus bis zu vier Farben bestehen. Welcher Teil eines Zeichens in welcher Farbe dargestellt wird, wird dabei durch jeweils zwei Bits der Zeichenmatrix bestimmt. Dazu muß man wissen, daß jedes Zeichen aus 8 mal 8 Punkten aufgebaut ist, die in jeweils acht Bytes codiert sind (mehr dazu im Abschnitt über Zeichensätze). Das "A" beispielsweise sieht so aus:

```
...**... —> 00011000   00 01 10 00
..****.. —> 00111100   00 11 11 00
.**  **. —> 01100110   01 10 01 10
.******. —> 01111110   01 11 11 10
.**..**. —> 01100110   01 10 01 10
.**..**. —> 01100110   01 10 01 10
.**..**. —> 01100110   01 10 01 10
........ —> 00000000   00 00 00 00
```

In der Abbildung links sehen Sie, wie die Bits zu Paaren zusammengefaßt werden. Jede der vier Kombinationsmöglichkeiten dient als Zeiger auf ein Farb-Register:

Matrix-Bits	Farb-Register
0 0	53281
0 1	53282
1 0	53283
1 1	Farbspeicher (Bits 0-2)

Die ersten drei Farben werden also global für alle auf dem Bildschirm befindlichen Zeichen gewählt, während man die Farbe zu der Bit-Kombination "11" für jedes einzelne Zeichen bzw. jede Bildschirmposition festlegen kann. Allerdings hat man nur die ersten acht Farben zur Verfügung, da nur die Bits 0 bis 2 der Farbspeicherzellen berücksichtigt werden. Um den Multicolor-Modus einzuschalten, geben Sie ein:

```
POKE 53270,PEEK(53270) OR 16
```

Mit

```
POKE 53270,PEEK(53270) AND 239
```

gelangen Sie wieder in die Normaldarstellung. Mit Hilfe des folgenden Programms können Sie ein wenig experimentieren:

```
100 input "farbe 1:";f1
110 input "farbe 2:";f2
120 input "farbe 3:";f3
130 input "farbe 4:";f4
140 input "text:"tx$
150 print chr$(147);tx$
160 :
170 rem farben setzen
180 poke 53281,f1
190 poke 53282,f2
200 poke 53283,f3
210 for z=0 to len(tx$)-1
220 poke 55296+z,f4
230 next z
240 :
250 rem multicolor-modus ein
260 poke 53270,peek(53270) or 16
270 :
280 get eg$:if eg$="" then 280
290 :
300 rem multicolor-modus aus
310 poke 53270,peek(53270) and 239
320 :
330 input "noch einmal: (j/n) ";jn$
340 if jn$="j" then run
```

Wie Sie gesehen haben, sind beide Farbmodi mit starken Einschränkungen im Hinblick auf die Anzahl bzw. die Lesbarkeit der Zeichen verbunden. Der Erweiterte-Hintergrund-Farbmodus und der Multicolor-Modus werden daher in Programmen nur recht selten eingesetzt.

6.2 Hochauflösende Grafik

In der hochauflösenden Grafik haben Sie eine Auflösung von 64000 Einzelpunkten, das sind 64-mal mehr als auf dem Textbildschirm. In dieser Auflösung stehen jeweils zwei Farben zur Verfügung. Eine für die gesetzten und eine für die gelöschten Punkte.

Wie bereits erwähnt, gibt es einen zweiten Modus, in dem man mit jeweils vier Farben arbeiten kann. Da sich die Auflösung dabei aber auf nur noch 160*200 Punkte verringert, kann man die Grafik fast schon nicht mehr als "hochauflösend" bezeichnen. Ich werde daher im folgenden nur auf den Zweifarbmodus mit 320*200 Punkten eingehen.

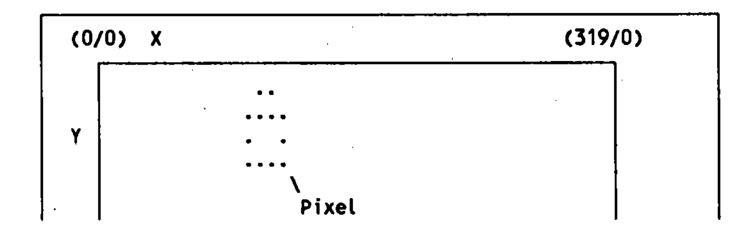
6.2.1 Grundlagen

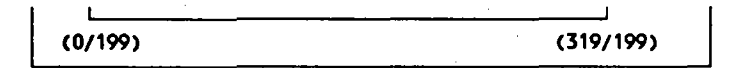
Ähnlich wie der Textbildschirm mit seinem Bildschirmspeicher benötigt auch die hochauflösende Grafik einen Speicherbereich, in dem vermerkt ist, welche Punkte gesetzt und welche gelöscht sind.

Diesen Speicherbereich bezeichnet man als "Bitmap" der hochauflösenden Grafik, da zur Codierung eines Punktes gerade ein Bit benötigt wird (ist ein Punkt gelöscht, dann ist sein zugehöriges Bit gleich 0 ist der Punkt gesetzt, hat das Bit den Wert 1). Da die Grafik ja aus 64.000 Einzelpunkten besteht, bedarf es 64.000 Bits oder 8.000 Bytes an Speicherplatz für eine Hires-Bitmap.

Das ist nicht gerade wenig! Man muß sich daher sehr genau überlegen, wo im Speicher die Bitmap abgelegt werden soll. Das Entscheidende aber ist das Setzen eines einzelnes Punktes, auch "Pixel" genannt, der hochauflösenden Grafik. Eine Routine, die diese Aufgabe übernimmt, ist der zentrale Bestandteil jeden Grafikpakets.

Die Grafik wird dazu in ein X/Y-Koordinatensystem eingeteilt. Die folgende Abbildung verdeutlicht die Zusammenhänge:





Die linke obere Ecke hat also die Koordinaten 0/0. In horizontaler Richtung werden die X-Koordinaten abgetragen (von 0 bis 319), in vertikaler Richtung die Y-Koordinaten (von 0 bis 199).

6.2.2 Die hochauflösende Grafik programmieren

Wie schaltet man die hochauflösende Grafik nun aber ein? Wie legt man die Bitmap fest? Und wie setzt oder löscht man ein Pixel der Grafik? All das gestaltet sich in BASIC sehr mühsam und vor allem sehr zeitaufwendig. Ich habe Ihnen daher wieder einige Maschinensprache-Routinen geschrieben, die im folgenden abgedruckt sind. Hinter dem Assembler-Listing finden Sie den gewohnten BASIC-Lader zu allen Routinen.

Außerdem benötigen wir vier Routinen aus Kapitel 4: TRANSFER, MYFILL, DSAVEM und DLOADM. Falls Sie diese Routinen noch nicht abgetippt haben, sollten Sie das jetzt schnell nachholen. Das Assembler-Listing von HIRESGRAFIK:

```
100 ;***********************************************************
105 ;*                                                         *
110 ;* programm: hiresgrafik                                   *
120 ;* routinen zur hiresgrafikprogrammierung                  *
140 ;*                                                         *
150 ;***********************************************************
200 ;
205 ;
210 .ba 50465    ;*** startadresse ***
220 ;
230 ;
240 ;*** labels ***********************************************
245 .gl  zw =   2   ;zwischenspeicher
250 .gl zp1 = 251
255 .gl zp2 = 252
260 .gl zp3 = 253
265 .gl zp4 = 254
270 ;
280 ;
290 ;***********************************************************
295 ;*                                                         *
300 ;* hrmappos - legt die position der bitmap der             *
```

```
305 ;*              hochaufloesenden grafik fest                 *
310 ;*                                                           *
315 ;* aufruf: sys 50465,br                                      *
320 ;*     br: bitmap-bereich (0 = 57344-65343                   *
325 ;*                          1 =  8192-16191)                 *
330 ;*                                                           *
335 ;*************************************************************
340 ;
370 hrmappos   jsr $aefd      ;auf komma testen
380            jsr $b79e      ;bereichsnummer holen
390            cpx #2         ;>1?
400            bcc hrm1       ;nein
410            jmp $b248      ;ja, 'illegal quantity'
420 hrm1       stx hrmpps
430            lda #0         ;bitmap-basisadresse (low)
440            sta hrmpl
450            lda hrtab2,x   ;highwert
460            sta hrmph
470            rts            ;fertig!
```

Zunächst müssen Sie mit HRMAPPOS die Lage der Hiresbitmap im Rechnerspeicher festlegen. Dabei stehen Ihnen zwei Bereiche zur Auswahl.

Der Bereich von Speicherzelle 57.344 bis 65.343 hat den großen Vorteil, daß er den BASIC-Speicher nicht beschneidet. Wenn Sie den Bereich von 8.192 bis 16.191 wählen, stehen Ihnen nur noch etwa 6.000 Byte für BASIC zur Verfügung. Möchte man nur mit einer Bitmap arbeiten, sollte man daher immer den Bereich 0 wählen, der auch voreingestellt ist.

In vielen Fällen ist es aber durchaus sinnvoll zwei Bitmaps zur Verfügung zu haben. Während die eine Bitmap auf dem Bildschirm dargestellt wird, zeichnet man auf der zweiten Bitmap eine neue Grafik. Sobald diese fertiggestellt ist, schaltet man blitzschnell auf die zweite Bitmap um. Häufig möchte man auch Teile einer Grafik in eine andere kopieren. Zu diesem Zweck lädt man die Grafiken in die beiden Bitmaps und verschiebt den betreffenden Bereich mit Hilfe der Routine TRANSFER. Mehr dazu etwas weiter unten.

```
480 ;
490 ;
495 ;*************************************************************
500 ;*                                                           *
505 ;* hrcol - legt die farben fuer die hochauf-                 *
```

```
510 ;*          loesende grafik fest                           *
515 ;*                                                         *
520 ;* aufruf: sys 50493,gd,pt                                 *
525 ;*     gd: hintergrundfarbe (0-15)                         *
530 ;*     pt: punktfarbe (0-15)                               *
535 ;*                                                         *
540 ;**********************************************************
545 ;
590 hrcol       jsr $aefd         ;auf komma testen
600             jsr $b79e         ;hintergrundfarbe holen
610             stx zw
620             jsr $aefd         ;auf komma testen
630             jsr $b79e         ;punktfarbe holen
640             txa               ;farbcode
650             asl               ;*16
660             asl
670             asl
680             asl
690             ora zw            ;mit hintergrundfarbe
700             sta zw            ;verknuepfen
710             lda #0            ;bildschirmspeicherbasis (low)
720             sta zp1
730             ldy hrmpps
740             lda hrtab1c,y     ;(high)
750             sta zp2
760             ldx #4            ;anzahl pages
770             ldy #0            ;farbe setzen
780             lda zw            ;farbcode
790 hrc1        sta (zp1),y
800             iny
810             bne hrc1
820             inc zp2           ;page+1
830             dex
840             bne hrc1
850             rts               ;fertig!
```

Für die gelöschten und die gesetzten Punkte der Grafik können Sie jeweils eine Farbe auswählen. Wenn man weiß, wie und wo die Farben gespeichert werden, kann man jedoch leicht Grafiken mit allen 16 Farben entwerfen!

Die Farbcodes werden nämlich im Bildschirmspeicher des Textbildschirms abgelegt. Der Code für die Hintergrundfarbe in den unteren vier Bits jeder Speicherzelle, der Code für die Punktfarbe in den oberen vier Bits. Jede Speicherzelle ist für einen 8 mal 8 Punkte großen Bereich der Hires-Grafik zuständig. Wenn Sie nun mit HRCOL die Farben festlegen, werden einfach alle 1000 Speicherzellen mit demselben Farbcode belegt.

Es steht Ihnen aber natürlich nichts im Wege mittels POKE den Wert jeder Speicherzelle (und damit die Farben des zugehörigen Hires-Bereichs) individuell zu setzen. Der Bildschirmspeicher des Hires-Bereichs 0 erstreckt sich von Speicherzelle 52.224 bis 53.223, der Speicher des Bereichs 1 von Adresse 1.024 bis 2.023.

Wenn wir von der gewohnten Zeilen-/Spalteneinteilung ausgehen, können Sie die Farben für jeden 8 x 8-Punkte-Block leicht so einstellen:

```
130 ze=0:rem zeile (0-24)
140 sp=0:rem spalte (0-39)
150 :
160 pt=5:rem punktfarbe
170 gd=6:rem hintergrundfarbe
180 :
190 poke 52224+40*ze+sp,pt*16+gd:rem bereich 0
200 poke 1024+40*ze+sp,pt*16+gd:rem bereich 1
```

In der Regel werden Sie mit zwei Farben für die gesamte Grafik aber ganz gut auskommen. In diesem Fall genügt dann natürlich die Verwendung von HRCOL. Kommen wir zum Ein- und Ausschalten der Hires-Grafik:

```
860 ;
870 ;
875 ;**********************************************************
880 ;*                                                        *
885 ;* hron - schaltet die hochaufloesende grafik ein         *
890 ;*                                                        *
895 ;* aufruf: sys 50545,br                                   *
900 ;*     br: bitmap-bereich (0 = 57344-65343                *
905 ;*                         1 =  8192-16191)               *
910 ;*                                                        *
915 ;**********************************************************
920 ;
960 hron      jsr $aefd     ;auf komma testen
970           jsr $b79e     ;bereichsnummer holen
980           cpx #2        ;>1?
990           bcc hro1      ;nein
1000          jmp $b248     ;ja, 'illegal quantity'
1010 hro1     lda 53272     ;vic-register
1020          and #2
1030          ora hrtab1a,x ;einstellwert1 (bitmap-Position)
1040          sta 53272
1050          lda 56576     ;cia-register
1060          and #252
1070          ora hrtab1b,x ;einstellwert2 (16kbyte-Bereich)
```

```
1080           sta 56576
1090           lda 53265      ;vic-register
1100           ora #32        ;bit5 setzen (=hires ein)
1110           sta 53265
1120           rts            ;fertig!
1130 ;
1140 ;
1145 ;*****************************************************
1150 ;*                                                   *
1155 ;* hroff - schaltet die hochaufloesende grafik aus   *
1160 ;*                                                   *
1165 ;* aufruf: sys 50589                                 *
1170 ;*                                                   *
1175 ;*****************************************************
1180 ;
1210 hroff     lda 53265      ;vic-register
1220           and #223       ;bit5 loeschen (=hires aus)
1230           sta 53265
1240           lda 53272
1250           and #2
1260           ora #21
1270           sta 53272
1280           lda 56576      ;cia-register
1290           and #252
1300           ora #3         ;16kbyte-bereich
1310           sta 56576
1320           lda #4         ;page-wert des bildschirms
1330           sta 648
1340           jmp $e566      ;cursor home durchfuehren
```

Die Hiresgrafik wird aber erst durch HRON eingeschaltet. Mit HRMAPPOS wird nur die Bitmap für die folgenden Zeichenroutinen festgelegt. Wie schon erwähnt, kann man daher leicht die eine Bitmap darstellen lassen, während in der anderen gerade eine neue Grafik gezeichnet wird. Wenn Sie die Grafik mit HRON eingeschaltet haben, wird sich Ihnen höchstwahrscheinlich ein recht chaotisches Grafikmuster am Bildchirm präsentieren. Das liegt daran, daß ja der Bitmap-Speicher im Augenblick irgendwelche undefinierten Werte enthält.

Wir müssen die Bitmap erst einmal löschen. Dazu verwenden wir die Routine MYFILL zum Auffüllen von Speicherbereichen mit einem Wert, hier dem Wert Null:

```
SYS 49402,57344,65343,0:REM BEREICH 0
SYS 49402,8192,16191,0:REM BEREICH 1
```

Jetzt sind alle Vorbereitungen getroffen, um mit dem Zeichnen beginnen zu können:

```
1350 ;
1360 ;
1365 ;*******************************************************
1370 ;*                                                     *
1375 ;* hrplot - setzt, loescht oder invertiert einen       *
1380 ;*          punkt der hires-grafik                     *
1385 ;*                                                     *
1390 ;* aufruf: sys 50625,x,y,zm                            *
1395 ;*        x: x-koordinate (0-319)                      *
1400 ;*        y: y-koordinate (0-199)                      *
1405 ;*       zm: zeichenmodus (0 = setzen, 1 = loeschen,   *
1410 ;*                         2 = invertieren)            *
1415 ;*                                                     *
1420 ;*******************************************************
1425 ;
1430 hrplot    jsr $aefd      ;auf komma testen
1440           jsr coord      ;koordinaten holen
1450           jsr $aefd      ;auf komma testen
1460           jsr $b79e      ;zeichenmodus holen
1470           cpx #3         ;>2?
1480           bcc hrp1       ;nein, ok
1490           jmp $b248      ;ja, 'illegal quantity'
1500 hrp1      stx hrdrdm     ;zeichenmodus speichern
1510           sei            ;auf ram umschalten
1520           lda #53
1530           sta $01
1540           jsr plot       ;zur plot-routine
1550           lda #55        ;auf rom zurueckschalten
1560           sta $01
1570           cli
1580           rts            ;fertig!
1590 ;
1600 ;
1605 ;*******************************************************
1610 ;*                                                     *
1615 ;* hrtestp - testet, ob ein einzelner punkt der        *
1620 ;*           hires-grafik gesetzt ist                  *
1625 ;*                                                     *
1630 ;* aufruf: sys 50661,x,y                               *
1635 ;*        x: x-koordinate (0-319)                      *
1640 ;*        y: y-koordinate (0-199)                      *
1645 ;*        peek(251)=0: punkt nicht gesetzt             *
1650 ;*        peek(251)=1: punkt gesetzt                   *
1655 ;*                                                     *
1660 ;*******************************************************
1665 ;
1690 hrtestp   jsr $aefd      ;auf komma testen
1700           jsr coord      ;koordinaten holen
1710           ldx #3         ;zeichenmodus=testen
```

```
1720          stx hrdrdm
1730          sei           ;auf ram umschalten
1740          lda #53
1750          sta $01
1760          jsr plot      ;zur plot-routine
1770          lda #55       ;auf rom zurueckschalten
1780          sta $01
1790          cli
1800          sty zp1       ;code ablegen
1810          rts           ;fertig!
```

Der Aufruf von HRPLOT ist denkbar einfach. Neben den X- und Y-Koordinaten des betreffenden Punktes müssen Sie nur noch festlegen, ob der Punkt gesetzt oder gelöscht werden soll. Falls Sie den Zeichenmodus 2 wählen, wird der Punkt invertiert, d.h., wenn er gesetzt ist, wird er gelöscht und umgekehrt.

Häufig möchte man den momentanen "Zustand" (gesetzt oder gelöscht) eines bestimmten Pixels erfahren. Zu diesem Zweck gibt es die Routine HRTESTP, der dann die Koordinaten des zu testenden Punktes übergeben werden. Mit Hilfe von HRPLOT läßt sich im Prinzip jedes beliebige grafische Objekt erzeugen. Das Zeichnen von geraden Linien benötigt man aber derart oft, daß ich dafür eine eigene Routine geschrieben habe:

```
1820 ;
1830 ;
1840 ;*******************************************************
1845 ;*                                                     *
1850 ;* hrline - zeichnet eine linie in die hires-grafik    *
1855 ;*                                                     *
1860 ;* aufruf: sys 50688,x1,y1,x2,y2,zm                    *
1865 ;*  x1,y1: koordinaten des anfangspunkts               *
1870 ;*  x2,y2: koordinaten des endpunkts                   *
1875 ;*     zm: zeichenmodus (0 = setzen, 1 = loeschen,     *
1880 ;*                       2 = invertieren)              *
1885 ;*                                                     *
1890 ;*******************************************************
1895 ;
1900 hrline   jsr $aefd     ;auf komma testen
1910          jsr coord     ;koord. holen (anfangspunkt)
1920          ldx #2        ;und zwischenspeichern
1930 hrln1    lda zp1,x
1940          sta $b7,x
1950          dex
1960          bpl hrln1
1970          jsr $aefd     ;auf komma testen
1980          jsr coord     ;koord. holen (endpunkt)
```

```
1990            jsr $aefd       ;auf komma testen
2000            jsr $b79e       ;zeichenmodus holen
2010            cpx #3          ;>2?
2020            bcc hrln2       ;nein, ok
2030            jmp $b248       ;ja, 'illegal quantity'
2040 hrln2      stx hrdrdm      ;zeichenmodus speichern
2050            sei             ;auf ram umschalten
2060            lda #53
2070            sta $01
2080            jsr line        ;zur line-routine
2090            lda #55         ;auf rom zurueckschalten
2100            sta $01
2110            cli
2120            rts             ;fertig!
SYS 50688,0,0,319,199,0
SYS 50688,199,0,319,0,0
```

Diese Routine beispielsweise zeichnet die beiden Bildschirmdiagonalen in die Grafik. Assembler-Programmierer vergessen bitte nicht, noch die folgenden Programmzeilen abzutippen:

```
2130 ;
2135 ;
2140 ;
2145 ;*****************************************************
2150 ;*                                                   *
2155 ;*   ausfuehrungsroutinen                            *
2160 ;*                                                   *
2165 ;*****************************************************
2170 ;
2175 ;
2180 ;*** plot-routine ***********************************
2190 plot       lda zp1         ;y-coord.
2200            lsr             ;/8
2210            lsr
2220            lsr
2230            tay
2240            clc
2250            lda zp1         ;y-coord.
2260            and #7
2270            adc hrtab3,y    ;+(320*y/8)
2280            sta $14         ;zwischenspeichern
2290            lda zp2         ;x-coord. (low)
2300            and #248
2310            adc $14
2320            sta $14
2330            lda hrmph       ;bitmap-basis
2340            adc hrtab4,y    ;+(320*y/8)
2350            adc zp3         ;x-coord. (high)
2360            sta $15
2370            lda zp2         ;x-coord. (low)
```

```
2380          and #7
2390          tay
2400          lda hrtab5,y   ;bit-position
2410          ldy #0
2420          ldx hrdrdm     ;zeichenmodus
2430          bne pt1
2440 ;*** zeichnen ***
2450          ora ($14),y    ;entspr. bit setzen
2460          sta ($14),y
2470          rts            ;fertig!
2480 pt1      dex
2490          bne pt2
2500 ;*** loeschen ***
2510          eor #255
2520          and ($14),y    ;entspr. bit loeschen
2530          sta ($14),y
2540          rts            ;fertig!
2550 pt2      dex
2560          bne pt3
2570 ;*** invertieren ***
2580          eor ($14),y    ;entspr. bit umdrehen
2590          sta ($14),y
2600          rts            ;fertig!
2610 pt3      dex
2620          bne pt5
2630 ;*** testen ***
2640          lda zp2        ;x-coord. (low)
2650          and #7
2660          eor #7         ;bits umdrehen
2670          tax            ;als zaehler nach x
2680          lda ($14),y    ;bitmap-byte holen
2690 pt4      cpx #0
2700          beq pt4a
2710          lsr
2720          dex
2730          jmp pt4
2740 pt4a     and #1         ;bits1-7 loeschen
2750          tay
2760          rts            ;fertig!
2770 pt5      sta zw         ;bit-position zwischenspeichern
2780          rts            ;fertig!
2790 ;
2810 ;*** berechnungstabellen ***
2830 hrtab3   .by 0,64,128,192
2840          .by 0,64,128,192
2850          .by 0,64,128,192
2860          .by 0,64,128,192
2870          .by 0,64,128,192
2880          .by 0,64,128,192
2890          .by 0
2910 hrtab4   .by 0,1,2,3
2920          .by 5,6,7,8
2930          .by 10,11,12,13
```

```
2940            .by 15,16,17,18
2950            .by 20,21,22,23
2960            .by 25,26,27,28
2970            .by 30
2990 hrtab5     .by 128,64,32,16
3000            .by 8,4,2,1
3010 ;
3020 ;
3040 ;*** line-routine ****************************************
3070 line       lda zp1         ;aktuelle punktkoord.
3080            pha
3090            lda zp2
3100            pha
3110            lda zp3
3120            pha
3130 ;*** differenzen berechnen (x-coord.) ***
3140            sec
3150            lda $b8         ;xlow (1.punkt)
3160            sbc zp2         ;xlow (2.punkt)
3170            pha
3180            lda $b9         ;xhigh (1.punkt)
3190            sbc zp3         ;xhigh (2.punkt)
3200            sta az
3210            bcs ln1
3220            pla
3230            eor #$ff
3240            adc #1
3250            pha
3260            lda #0
3270            sbc az
3280 ln1        sta az+2
3290            sta az+5
3300            pla
3310            sta az+1
3320            sta az+4
3330 ;*** differenzen berechnen (y-coord.) ***
3340            clc
3350            lda $b7         ;y (1. punkt)
3360            sbc zp1         ;y (2. punkt)
3370            bcc ln2
3380            eor #$ff
3390            adc #$fe
3400 ln2        sta az+3
3410            ror az
3420 ;*** zaehler berechnen ***
3430            sec
3440            sbc az+1
3450            sta zw
3460            lda #$ff
3470            sbc az+2
3480            sta zp4
3490            jsr plot        ;1. punkt setzen
3500            jmp ln3
```

```
3510 ;*** x (c=1) oder y (c=0) bearbeiten ***
3520 lnlp      bcc ln6
3540 ln5       jsr xdeinc      ;x-coord. +1/-1
3550 ln3       lda az+4
3560           adc az+3
3570           sta az+4
3580           lda az+5
3590           sbc #0
3600           jmp ln4
3620 ln6       jsr ydeinc      ;y-coord. +1/-1
3630           clc
3640           lda az+4
3650           adc az+1
3660           sta az+4
3670           lda az+5
3680           adc az+2
3690 ln4       sta az+5
3700           php
3710           jsr plot        ;punkt setzen
3720           plp
3730           inc zw          ;zaehler
3740           bne lnlp
3750           inc zp4
3760           bne lnlp
3770 ;*** aktuelle punktkoord. zurueckholen ***
3780           pla
3790           sta zp3
3800           pla
3810           sta zp2
3820           pla
3830           sta zp1
3840           rts             ;fertig!
3850 ;
3870 ;*** x-coord. +1/-1 ***
3890 xdeinc    bit az          ;flag
3900           bxdc1        ;bit6 nicht gesetzt, dann inc.
3920           lda zp2         ;xlow (2. punkt)
3930           ora zp3         ;xhigh (2. punkt)
3940           beq xic1        ;x=0, dann fertig
3950           lda zp2         ;x-coord. -1
3960           bne xic2
3970           dec zp3         ;xhigh
3980 xic2      dec zp2         ;xlow
3990           sec
4000           rts             ;fertig!
4020 xdc1      lda zp3         ;xhigh
4030           beq xdc2        ;=0, ok
4040           lda zp2         ;xlow
4050           cmp #63         ;=319?
4060           beq xic1        ;ja, fertig
4070 xdc2      inc zp2         ;x-coord. +1
4080           bne xdc3
4090           inc zp3
```

```
4100 xdc3     sec
4110          rts           ;fertig!
4120 xic1     clc
4130          rts           ;fertig!
4140 ;
4170 ;*** y-coord. +1/-1 ***
4190 ydeinc   bit az        ;flag
4200          bmi ydc1      ;bit7 nicht gesetzt, dann inc.
4220          lda zp1       ;y-coord.
4230          beq xic1      ;=0, dann fertig
4240          dec zp1       ;y-coord. -1
4250          sec
4260          rts           ;fertig!
4280 ydc1     lda zp1       ;y-coord.
4290          cmp #199      ;ende erreicht?
4300          beq xic1      ;ja, fertig
4310          inc zp1
4320          sec
4330          rts           ;fertig!
4340 ;
4370 ;*** punktkoord. holen und testen ***
4380 coord    jsr $b7eb     ;x- und y-koordinate holen
4390          cpx #200      ;y-koord. >199?
4400          bcc crd1      ;nein, ok
4410 crderr   jmp $b248     ;ja, fehler
4420 crd1     ldy $14       ;x-koord. (low)
4430          lda $15       ;x-koord. (high)
4440          cmp #>(320)   ;>319?
4450          bcc crd2      ;<, ok
4460          bne crderr    ;>, fehler
4470          cpy #<(320)   ;x-koord. (low) >319?
4480          bcs crderr    ;ja, fehler
4490 crd2     stx zp1       ;y-koord.
4500          sty zp2       ;x-koord. (low)
4510          sta zp3       ;(high)
4520          rts           ;fertig!
4530 ;
4540 ;
4550 ;
4560 ;*** tabellen *****************************************
4570 ;
4590 ;*** umschaltwerte ***
4600 hrtab1a  .by 56,24
4610 hrtab1b  .by 0,3
4620 hrtab1c  .by 204,4
4630 ;
4650 ;*** bitmap-basisadressen ***
4660 hrtab2   .by 224,32
4665 ;
4670 ;
4675 ;
4680 ;*** datenspeicher ************************************
4690 hrmpps   .by 0         ;bitmap-bereichsnummer
```

```
4700 hrmpl    .by 0          ;bitmap-basis (low)
4710 hrmph    .by 0          ;(high)
4720 hrdrdm   .by 0          ;zeichenmodus
4730 az       .by 0,0,0,0,0,0,0,0,0,0 ;zwischenspeicher
```

Und der BASIC-Lader:

```
100 rem ******************************************************
102 rem *                                                    *
104 rem * programm: hires-grafik                             *
106 rem * programmlaenge: 700 bytes                          *
108 rem * routinen zur hires-grafikprogrammierung            *
110 rem *                                                    *
112 rem ******************************************************
114 rem *                                                    *
116 rem * hrmappos - legt die position der bitmap der        *
118 rem *            hochaufloesenden grafik fest            *
120 rem *                                                    *
122 rem * aufruf: sys 50465,br                               *
124 rem *     br: bitmap-bereich (0 = 57344-65343            *
126 rem *                         1 =  8192-16191)           *
128 rem *                                                    *
130 rem ******************************************************
132 rem *                                                    *
134 rem * hrcol - legt die farben fuer die hochauf-          *
136 rem *         loesende grafik fest                       *
138 rem *                                                    *
140 rem * aufruf: sys 50493,gd,pt                            *
142 rem *     gd: hintergrundfarbe (0-15)                    *
144 rem *     pt: punktfarbe (0-15)                          *
146 rem *                                                    *
148 rem ******************************************************
150 rem *                                                    *
152 rem * hron - schaltet die hochaufloesende grafik ein     *
154 rem *                                                    *
156 rem * aufruf: sys 50545,br                               *
158 rem *     br: bitmap-bereich (0 = 57344-65343            *
160 rem *                         1 =  8192-16191)           *
162 rem *                                                    *
164 rem ******************************************************
166 rem *                                                    *
168 rem * hroff - schaltet die hochaufloesende grafik aus    *
170 rem *                                                    *
172 rem * aufruf: sys 50589                                  *
174 rem *                                                    *
176 rem ******************************************************
178 rem *                                                    *
180 rem * hrplot - setzt, loescht oder invertiert einen      *
182 rem *          punkt der hires-grafik                    *
184 rem *                                                    *
186 rem * aufruf: sys 50625,x,y,zm                           *
188 rem *      x: x-koordinate (0-319)                       *
```

```
190 rem *      y: y-koordinate (0-199)                         *
192 rem *     zm: zeichenmodus (0 = setzen, 1 = loeschen,      *
194 rem *                        2 = invertieren)              *
196 rem *                                                      *
198 rem ********************************************************
200 rem *                                                      *
202 rem * hrtestp - testet, ob ein einzelner punkt der         *
204 rem *           hires-grafik gesetzt ist                   *
206 rem *                                                      *
208 rem * aufruf: sys 50661,x,y                                *
210 rem *      x: x-koordinate (0-319)                         *
212 rem *      y: y-koordinate (0-199)                         *
214 rem *      peek(251)=0: punkt nicht gesetzt                *
216 rem *      peek(251)=1: punkt gesetzt                      *
218 rem *                                                      *
220 rem ********************************************************
222 rem *                                                      *
224 rem * hrline - zeichnet eine Linie in die hires-grafik *
226 rem *                                                      *
228 rem * aufruf: sys 50688,x1,y1,x2,y2,zm                     *
230 rem *  x1,y1: koordinaten des anfangspunkts                *
232 rem *  x2,y2: koordinaten des endpunkts                    *
234 rem *     zm: zeichenmodus (0 = setzen, 1 = loeschen,      *
236 rem *                        2 = invertieren)              *
238 rem *                                                      *
240 rem ********************************************************
280 :
290 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=50465:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 32,253,174,32,158,183,224,2,144,3,1205
1010 data 76,72,178,142,207,199,169,0,141,208,1392
1020 data 199,189,205,199,141,209,199,96,32,253,1722
1030 data 174,32,158,183,134,2,32,253,174,32,1174
1040 data 158,183,138,10,10,10,10,5,2,133,659
1050 data 2,169,0,133,251,172,207,199,185,203,1521
1060 data 199,133,252,162,4,160,0,165,2,145,1222
1070 data 251,200,208,251,230,252,202,208,246,96,2144
1080 data 32,253,174,32,158,183,224,2,144,3,1205
1090 data 76,72,178,173,24,208,41,2,29,199,1002
1100 data 199,141,24,208,173,0,221,41,252,29,1288
```

```
1110 data 201,199,141,0,221,173,17,208,9,32,1201
1120 data 141,17,208,96,173,17,208,41,223,141,1265
1130 data 17,208,173,24,208,41,2,9,21,141,844
1140 data 24,208,173,0,221,41,252,9,3,141,1072
1150 data 0,221,169,4,141,136,2,76,102,229,1080
1160 data 32,253,174,32,168,199,32,253,174,32,1349
1170 data 158,183,224,3,144,3,76,72,178,142,1183
1180 data 210,199,120,169,53,133,1,32,51,198,1166
1190 data 169,55,133,1,88,96,32,253,174,32,1033
1200 data 168,199,162,3,142,210,199,120,169,53,1425
1210 data 133,1,32,51,198,169,55,133,1,88,861
1220 data 132,251,96,32,253,174,32,168,199,162,1499
1230 data 2,181,251,149,183,202,16,249,32,253,1518
1240 data 174,32,168,199,32,253,174,32,158,183,1405
1250 data 224,3,144,3,76,72,178,142,210,199,1251
1260 data 120,169,53,133,1,32,209,198,169,55,1139
1270 data 133,1,88,96,165,251,74,74,74,168,1124
1280 data 24,165,251,41,7,121,151,198,133,20,1111
1290 data 165,252,41,248,101,20,133,20,173,209,1362
1300 data 199,121,176,198,101,253,133,21,165,252,1619
1310 data 41,7,168,185,201,198,160,0,174,210,1344
1320 data 199,208,5,17,20,145,20,96,202,208,1120
1330 data 7,73,255,49,20,145,20,96,202,208,1075
1340 data 5,81,20,145,20,96,202,208,22,165,964
1350 data 252,41,7,73,7,170,177,20,224,0,971
1360 data 240,5,74,202,76,135,198,41,1,168,1140
1370 data 96,133,2,96,0,64,128,192,0,64,775
1380 data 128,192,0,64,128,192,0,64,128,192,1088
1390 data 0,64,128,192,0,64,128,192,0,0,768
1400 data 1,2,3,5,6,7,8,10,11,12,65
1410 data 13,15,16,17,18,20,21,22,23,25,190
1420 data 26,27,28,30,128,64,32,16,8,4,363
1430 data 2,1,165,251,72,165,252,72,165,253,1398
1440 data 72,56,165,184,229,252,72,165,185,229,1609
1450 data 253,141,211,199,176,11,104,73,255,105,1528
1460 data 1,72,169,0,237,211,199,141,213,199,1442
1470 data 141,216,199,104,141,212,199,141,215,199,1767
1480 data 24,165,183,229,251,144,4,73,255,105,1433
1490 data 254,141,214,199,110,211,199,56,237,212,1833
1500 data 199,133,2,169,255,237,213,199,133,254,1794
1510 data 32,51,198,76,42,199,144,20,32,104,898
1520 data 199,173,215,199,109,214,199,141,215,199,1863
1530 data 173,216,199,233,0,76,78,199,32,145,1351
1540 data 199,24,173,215,199,109,212,199,141,215,1686
1550 data 199,173,216,199,109,213,199,141,216,199,1864
1560 data 8,32,51,198,40,230,2,208,203,230,1202
1570 data 254,208,199,104,133,253,104,133,252,104,1744
1580 data 133,251,96,44,211,199,80,16,165,252,1447
1590 data 5,253,240,28,165,252,208,2,198,253,1604
1600 data 198,252,56,96,165,253,240,6,165,252,1683
1610 data 201,63,240,8,230,252,208,2,230,253,1687
1620 data 56,96,24,96,44,211,199,48,8,165,947
1630 data 251,240,245,198,251,56,96,165,251,201,1954
```

```
1640 data 199,240,235,230,251,56,96,32,235,183,1757
1650 data 224,200,144,3,76,72,178,164,20,165,1246
1660 data 21,201,1,144,6,208,243,192,64,176,1256
1670 data 239,134,251,132,252,133,253,96,56,24,1570
1680 data 0,3,204,4,224,32,0,0,0,0,467
1690 data 0,0,0,0,0,0,0,0,0,0,0
2000 data -1:rem endmarkierung
```

Eine einmal erzeugte Grafik möchte man vielleicht dauerhaft auf Diskette sichern und später wieder einladen. Dazu bedienen wir uns der Routinen DSAVEM und DLOADM aus Kapitel 4:

```
SYS 49489,"GRAFIK0",57344,65343:REM BEREICH 0 SPEICHERN
SYS 49489,"GRAFIK1",8192,16191:REM BEREICH 1 SPEICHERN
SYS 49676,"GRAFIK0",57344:REM BEREICH 0 LADEN
SYS 49676,"GRAFIK1",8192:REM BEREICH 1 LADEN
```

Wie schon weiter oben erwähnt, kann man mit Hilfe der Routine TRANSFER leicht einzelne Grafikbereiche zwischen den beiden Bitmaps umspeichern, im Extremfall die gesamte Grafik:

```
SYS 49152,57344,65343,8192,3:REM BEREICH 0 IN BEREICH 1
SYS 49152,8192,16191,57344,3:REM BEREICH 1 IN BEREICH 0
```

Bei kleineren Bereichen möchte ich der Einfachheit halber davon ausgehen, daß sie sich aus 8 Punkt großen Zeilenblöcken zusammensetzen und entsprechend der Spalteneinteilung des Textbildschirms in der Grafik liegen. Um die X/Y-Koordinaten in die richtigen Speicheradressen umrechnen zu können, muß man wissen, daß die Grafik in der Bitmap zeilenweise gespeichert wird. Die erste Grafikzeile von 0/0 bis 319/0 liegt im Bitmap-Bereich 0, also von Speicherzelle 57.344 bis 57.383, die zweite Grafikzeile von Adresse 57.384 bis 57.423 usw.

Nehmen wir einmal an, Sie möchten aus dem Grafikbereich 0 einen rechteckigen Block mit den Eckkoordinaten 16/16 (linke obere Ecke) und 160/32 (rechte untere Ecke) an dieselbe Stelle des Bereichs 1 verschieben. Die Rechnung dazu sieht ganz allgemein wie folgt aus:

```
100 b0=57344:rem bitmap-basisadressen
110 b1=8192
120 :
130 xa=16:ya=16:rem koordinaten
```

```
140 xe=160:ye=32
145 :
150 pa=ya*40+xa:rem anfangsspalte
160 pe=ya*40+xe:rem endspalte
170 for z=0 to (ye-ya)-1:rem zeilenweise umspeichern
180 sys 49152,b0+pa+z*40,b0+pe+z*40,b1+pa+z*40,3:rem transfer
190 next z
```

6.2.3 Fortgeschrittene Techniken

Mit Hilfe der beiden Routinen HRPLOT und HRLINE können wir nun fast jedes beliebige andere Grafikobjekt zeichnen, beispielsweise Kreise und Rechtecke oder Ellipsen und Polygone. Das folgende Programm beispielsweise zeichnet beliebige Kreise und Ellipsen. Da ja der Kreis nur ein Sonderfall der Ellipse ist, genügt eine Routine.

Wollen Sie einen Kreis zeichnen, geben Sie für den horizontalen und vertikalen Radius einfach denselben Wert an:

```
70 rem kreise und ellipsen zeichnen
80 :
90 :
100 rem maschinenprogramme nachladen
110 if fl=0 then fl=1:load "hires-grafik",8,1
120 if fl=1 then fl=2:load "transfer",8,1
130 if fl=2 then fl=3:load "myfill",8,1
140 if fl=3 then fl=4:load "dsavem",8,1
150 if fl=4 then fl=5:load "dloadm",8,1
160 :
170 rem vorbereitungen
180 ba=57344:rem bitmap-basisadresse
190 br=0:rem bereichsnummer
200 sys 50465,br:rem bitmap-bereich festlegen (hrmappos)
210 sys 50493,0,1:rem farben (hrcol)
220 sys 49402,ba,ba+7999,0:rem bitmap loeschen (myfill)
230 zm=0:rem zeichenmodus
240 :
250 rem mittelpunkt und radien erfragen
260 input "mittelpunkt (x-koordinate): ";xm
270 input "mittelpunkt (y-koordinate): ";ym
280 input "horizontaler radius: ";hr
290 input "vertikaler radius: ";vr
300 :
310 rem grafik ein (hron)
320 sys 50545,br
330 :
```

```
340 rem ellipse/kreis zeichnen
350 for wg=0 to 360:rem winkelgrade
360 wb=wg*3.14/180:rem umrechnung in bogenmaß fuer sin/cos
370 x=xm+hr*cos(wb)
380 y=ym+vr*sin(wb)
390 sys 50625,x,y,zm:rem hrplot
400 next wg
410 :
420 rem auf tastendruck warten
430 get eg$:if eg$="" then 430
440 :
450 rem grafik aus (hroff)
460 sys 50589
470 :
480 print chr$(147)
490 input "noch einmal: (j/n)";jn$
500 if jn$="j" then 260
510 end
```

Durch Änderung der FOR-Anweisung in Programmzeile 350 können Sie übrigens auch leicht Ellipsenteile zeichnen lassen.

```
350 for wg=0 to 180:rem winkelgrade
```

beispielsweise zeichnet eine halbe Ellipse. Zum Zeichnen eines Rechtecks verwenden wir die Routine HRLINE. Dabei genügt die Angabe der linken oberen und der rechten unteren Ecke des Rechtecks:

```
70 rem rechtecke zeichnen
80 :
90 :
100 rem maschinenprogramme nachladen
110 if fl=0 then fl=1:load "hires-grafik",8,1
120 if fl=1 then fl=2:load "transfer",8,1
130 if fl=2 then fl=3:load "myfill",8,1
140 if fl=3 then fl=4:load "dsavem",8,1
150 if fl=4 then fl=5:load "dloadm",8,1
160 :
170 rem vorbereitungen
180 ba=57344:rem bitmap-basisadresse
190 br=0:rem bereichsnummer
200 sys 50465,br:rem bitmap-bereich festlegen (hrmappos)
210 sys 50493,0,1:rem farben (hrcol)
220 sys 49402,ba,ba+7999,0:rem bitmap loeschen (myfill)
230 zm=0:rem zeichenmodus
240 :
250 rem eckpunkte erfragen
260 input "linke obere ecke (x-koordinate): ";lx
270 input "linke obere ecke (y-koordinate): ";ly
```

```
280 input "rechte untere ecke (x-koordinate): ";rx
290 input "rechte untere ecke (y-koordinate): ";ry
300 :
310 rem grafik ein (hron)
320 sys 50545,br
330 :
340 rem rechteck zeichnen (hrline)
350 sys 50688,lx,ly,rx,ly,zm:rem oberer teil
360 sys 50688,rx,ly,rx,ry,zm:rem rechter teil
370 sys 50688,lx,ry,rx,ry,zm:rem unterer teil
380 sys 50688,lx,ly,lx,ry,zm:rem linker teil
410 :
420 rem auf tastendruck warten
430 get eg$:if eg$="" then 430
440 :
450 rem grafik aus (hroff)
460 sys 50589
470 :
480 print chr$(147)
490 input "noch einmal: (j/n)";jn$
500 if jn$="j" then 260
510 end
```

Wem vier Ecken zu wenig sind, sollte einmal das folgende Programm ausprobieren. Es zeichnet Polygonzüge beliebiger Größe. Die Eckkoordinaten werden dazu in DATA-Zeilen abgelegt:

```
70 rem polygone zeichnen
80 :
90 :
100 rem maschinenprogramme nachladen
110 if fl=0 then fl=1:load "hires-grafik",8,1
120 if fl=1 then fl=2:load "transfer",8,1
130 if fl=2 then fl=3:load "myfill",8,1
140 if fl=3 then fl=4:load "dsavem",8,1
150 if fl=4 then fl=5:load "dloadm",8,1
160 :
170 rem vorbereitungen
180 ba=57344:rem bitmap-basisadresse
190 br=0:rem bereichsnummer
200 sys 50465,br:rem bitmap-bereich festlegen (hrmappos)
210 sys 50493,0,1:rem farben (hrcol)
220 sys 49402,ba,ba+7999,0:rem bitmap loeschen (myfill)
230 zm=0:rem zeichenmodus
240 :
310 rem grafik ein (hron)
320 sys 50545,br
330 :
340 rem polygon zeichnen
350 read xa,ya:rem startwert
355 read xe,ye
```

```
360 if xe=-1 then 430
365 sys 50688,xa,ya,xe,ye,zm:rem (hrline)
370 xa=xe:ya=ye
380 goto 355
410 :
420 rem auf tastendruck warten
430 get eg$:if eg$="" then 430
440 :
450 rem grafik aus (hroff)
460 sys 50589
510 end
520 :
530 :
590 rem eckkoordinaten des polygons
600 data 10,10
610 data 100,50
600 data 300,180
610 data 200,140
600 data 50,190
610 data 80,100
600 data 10,10
610 data -1,-1:rem endmarkierung
```

Sehr beliebt ist auch die grafische Darstellung von Funktionsverläufen. Am Beispiel einer Sinuskurve möchte ich Ihnen zeigen, wie so etwas aussehen kann:

```
70 rem funktionsplotter
80 :
90 :
100 rem maschinenprogramme nachladen
110 if fl=0 then fl=1:load "hires-grafik",8,1
120 if fl=1 then fl=2:load "transfer",8,1
130 if fl=2 then fl=3:load "myfill",8,1
140 if fl=3 then fl=4:load "dsavem",8,1
150 if fl=4 then fl=5:load "dloadm",8,1
160 :
170 rem vorbereitungen
180 ba=57344:rem bitmap-basisadresse
190 br=0:rem bereichsnummer
200 sys 50465,br:rem bitmap-bereich festlegen (hrmappos)
210 sys 50493,0,1:rem farben (hrcol)
220 sys 49402,ba,ba+7999,0:rem bitmap loeschen (myfill)
230 zm=0:rem zeichenmodus
240 :
310 rem grafik ein (hron)
320 sys 50545,br
330 :
340 rem x-achse zeichnen
350 sys 50688,10,100,310,100,zm:rem (hrline)
355 :
```

```
360 rem y-achse zeichnen
370 sys 50688,160,10,160,190,zm:rem (hrline)
375 :
380 rem sinuskurve
390 for x=10 to 310
400 y=int(100-100*sin(x*3.14/180))
405 sys 50625,x,y,zm:rem hrplot
410 next x
415 :
420 rem auf tastendruck warten
430 get eg$:if eg$="" then 430
440 :
450 rem grafik aus (hroff)
460 sys 50589
470 end
```

Zum Schluß noch ein Programm, das einen sogenannten Moiree-Effekt erzeugt. Lassen Sie sich überraschen!

```
70 rem moiree-effekt
80 :
90 :
100 rem maschinenprogramme nachladen
110 if fl=0 then fl=1:load "hires-grafik",8,1
120 if fl=1 then fl=2:load "transfer",8,1
130 if fl=2 then fl=3:load "myfill",8,1
140 if fl=3 then fl=4:load "dsavem",8,1
150 if fl=4 then fl=5:load "dloadm",8,1
160 :
170 rem vorbereitungen
180 ba=57344:rem bitmap-basisadresse
190 br=0:rem bereichsnummer
200 sys 50465,br:rem bitmap-bereich festlegen (hrmappos)
210 sys 50493,0,1:rem farben (hrcol)
220 sys 49402,ba,ba+7999,0:rem bitmap loeschen (myfill)
230 zm=0:rem zeichenmodus
240 :
250 rem grafik ein (hron)
260 sys 50545,br
270 :
280 rem grafik zeichnen
290 x1=0:y1=0
290 for z=0 to 199 step 10
300 x2=z:y2=199
310 sys 50688,x1,y1,x2,y2,zm:rem (hrline)
320 x2=199:y2=199-z
330 sys 50688,x1,y1,x2,y2,zm:rem (hrline)
340 next z
350 x1=199:y1=199
360 for z=0 to 199 step 10
370 x2=0:y2=199-z
380 sys 50688,x1,y1,x2,y2,zm:rem (hrline)
```

```
390 x2=z:y2=0
400 sys 50688,x1,y1,x2,y2,zm:rem (hrline)
410 next z
415 :
420 rem auf tastendruck warten
430 get eg$:if eg$="" then 430
440 :
450 rem grafik aus (hroff)
460 sys 50589
510 end
```

6.3 Sprites

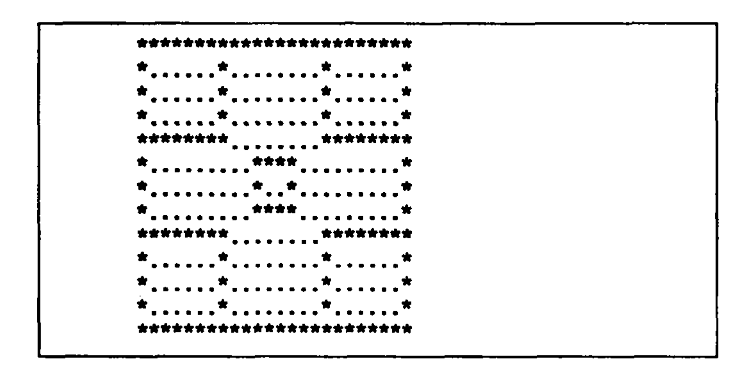
Sprites sind kleine Grafikobjekte, die unabhängig von der Text- und Grafikdarstellung über den Bildschirm bewegt werden können. Bis zu acht Sprites kann der Commodore 64 gleichzeitig darstellen. All das, was sich in Spielen an Raumschiffen, Männchen und sonstigen Spielfiguren am Bildschirm tummelt, sind in der Regel nichts anderes als Sprites. Aber auch der Mauszeiger von GEOS wird durch ein Sprite dargestellt. Sprites lassen sich also durchaus auch in "ernsthaften" Programmen sinnvoll einsetzen.

6.3.1 Grundlagen

Ein einzelnes Sprite ist aus genau 24 mal 21 Einzelpunkten aufgebaut, 24 Punkte in der Horizontalen und 21 Punkte in der Vertikalen. Um ein Sprite zu entwerfen, benutzen Sie am besten das folgende Punktraster:

```
........................
........................     S p r i t e -
........................
........................     e n t w u r f s -
........................
........................     r a s t e r
........................
........................
........................
........................
........................
```

Jeder Punkt des Rasters entspricht einem Grafikpunkt der gesetzt oder gelöscht sein kann. Ein einfaches Quadratmuster-Sprite könnte beispielsweise so aussehen:



Wenn Sie das Sprite so vielleicht auf ein Blatt Papier gezeichnet haben, kann der Commodore 64 natürlich nichts damit anfangen. Wir müssen die Sprite-Darstellung für ihn mit Zahlen codieren.

Dazu werden jeweils acht Punkte einer Zeile zu einem Byte zusammengefaßt. Jedes der acht Bits dieses Bytes repräsentiert einen Grafikpunkt. Ist der betreffende Grafikpunkt gelöscht, wird das zugehörige Bit auf Null gesetzt ist der Grafikpunkt gesetzt, erhält das Bit den Wert eins.

Da eine Sprite-Zeile aus 24 Punkten besteht, benötigt man für jede Zeile 3 Byte zur Codierung, für das gesamte Sprite also 3 mal 21 (Zeilen) = 63 Byte. Diese 63 Byte "von Hand" zu berechnen ist sehr mühsam. Etwas weiter unten finden Sie deshalb einen komfortablen Sprite-Editor, der Ihnen diese Arbeit komplett abnimmt.

Die Codezahlen des Sprites müssen nun in einem Speicherbereich des Commodore 64 abgelegt werden, dessen Anfangsadresse durch 64 teilbar ist. Wenn man mit dem normalen Textbildschirm arbeitet, verwendet man dazu am besten den sogenannten

"Kassettenpuffer" von Speicheradresse 828 bis 1.023. Dort lassen sich in den Bereichen 832-895, 896-959 und 960-1.023 bis zu drei Sprites unterbringen.

Sie dürfen aber auch einen anderen Bereich wählen, etwa "hinter" dem geladenen BASIC-Programm, falls dieses nicht zu lang ist. Hauptsache, die Startadresse des Speicherbereichs ist durch 64 teilbar. Damit der Commodore 64 weiß, wo Sie die Daten abgelegt haben, gibt es zu jedem der acht Sprites einen Zeiger, den man auf den Beginn des Speicherbereichs setzen muß. Diese Zeiger befinden sich hinter dem Bildschirmspeicher von Speicheradresse 2.040 bis 2.047. Die Zuordnung zu den Sprites ist wie folgt:

```
Sprite 1: 2040
Sprite 2: 2041
Sprite 3: 2042
Sprite 4: 2043
Sprite 5: 2044
Sprite 6: 2045
Sprite 7: 2046
Sprite 8: 2047
```

Nehmen wir einmal an, die Sprite-Daten wurden ab Speicheradresse 832 abgelegt und Sie möchten, daß das Sprite 5 in dieser Darstellung erscheinen soll. 832 geteilt durch 64 ergibt 13. Also schreiben wir in die Speicherzelle 2.044 den Wert 13. Nun weiß der Rechner also, wie das Sprite aussehen soll. Den kompliziertesten Teil des Ganzen haben wir damit schon hinter uns.

Um das Sprite jetzt einzuschalten, es an eine bestimmte Bildschirmposition zu setzen oder ihm eine bestimmte Farbe zu geben, gibt es Register im VIC-II-Chip, die man mit entsprechenden Werten belegen muß.

Keine Angst, Sie werden sich auch bei der Sprite-Programmierung nicht mit irgendwelchen Register-Adressen oder -positionen herumschlagen müssen. Ich habe für Sie wieder eine kleine Routinensammlung geschrieben, die alle Aspekte der Sprite-Programmierung abdeckt. Bei allen Routinen genügt die Angabe der Nummer (1-8) des anzusprechenden Sprites.

```
390            jsr $b79e      ;spalte einlesen
400            txa
410            beq crs1       ;=0, dann fehler
420            cpx #41        ;spalte > 40?
430            bcs crs1       ;ja, fehler
440            tay
450            dey
460            ldx zw         ;zeile
470            dex
480            clc            ;flag fuer 'cursor setzen'
490            jmp $fff0      ;cursor setzen
500 ;
502 ;
504 ;*********************************************************
506 ;*                                                       *
508 ;* cron/off - cursor ein-/ausschalten                    *
510 ;*                                                       *
512 ;* aufruf: sys 49974 (ein)                               *
514 ;*         sys 49979 (aus)                               *
516 ;*                                                       *
518 ;*********************************************************
520 ;
580 cron       lda #0         ;cursor einschalten
590            sta $cc
600            rts            ;fertig!
610 ;
620 croff      lda #1         ;cursor ausschalten
630            sta $cc
640            sei
650            lda $cf        ;cursor in blinkphase?
660            beq crf1       ;nein
670            lda $ce        ;zeichen unter dem cursor
680            ldx $0287      ;farbe unter dem cursor
690            ldy #00
700            sty $cf        ;blinkphase aus
710            jsr $ea13      ;zeichen und farbe setzen
720 crf1       cli
730            rts            ;fertig!
```

Und der BASIC-Lader:

```
100 rem *****************************************************
102 rem *                                                   *
104 rem * programm: cursor                                  *
106 rem * programmlaenge: 68 bytes                          *
108 rem * routinen zur cursor-steuerung                     *
110 rem *                                                   *
112 rem *****************************************************
114 rem *                                                   *
116 rem * crset - cursor positionieren                      *
118 rem *                                                   *
120 rem * aufruf: sys 49934,ze,sp                           *
```

```
122 rem *     ze: zeile (1-25)                                  *
124 rem *     sp: spalte (1-40)                                 *
126 rem *                                                       *
128 rem *********************************************************
130 rem *                                                       *
132 rem * cron/off - cursor ein-/ausschalten                    *
134 rem *                                                       *
136 rem * aufruf: sys 49974 (ein)                               *
138 rem *         sys 49979 (aus)                               *
140 rem *                                                       *
142 rem *********************************************************
280 :
290 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=49934:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 32,253,174,32,158,183,138,208,3,76,1257
1010 data 72,178,224,26,176,249,134,2,32,253,1346
1020 data 174,32,158,183,138,240,238,224,41,176,1604
1030 data 234,168,136,166,2,202,24,76,240,255,1503
1040 data 169,0,133,204,96,169,1,133,204,120,1229
1050 data 165,207,240,12,165,206,174,135,2,160,1466
1060 data 0,132,207,32,19,234,88,96,0,0,808
2000 data -1:rem endmarkierung
```

Haben Sie das Programm abgetippt und die Routinen als Maschinenprogramm auf der Diskette gespeichert? Gut, denn nur so können Sie den Sprite-Editor nutzen.

Die Bedienung des Sprite-Editors ist schnell erklärt. Die 63 Byte umfassenden Daten eines Sprites werden jeweils in der String-Variablen MT$ abgelegt. Von dort können sie später leicht weiterverarbeitet werden. Außerdem haben Sie dadurch die Möglichkeit, ein einmal entworfenes und berechnetes Sprite jederzeit wieder abändern zu können.

Daraus ergeben sich auch die beiden Aufrufvarianten für den Editor, der als Unterprogramm gestaltet ist.

In den meisten Fällen werden Sie ein neues Sprite entwerfen wollen. Dazu rufen Sie den Editor (vom Hauptprogramm aus) mit einem

```
GOSUB 1250
```

auf. Die berechneten Sprite-Daten befinden sich anschließend in der Variablen MT$. Wenn Sie diese Daten vielleicht später einmal als Grundlage für einen anderen Sprite-Entwurf benutzen wollen, empfiehlt es sich, sie in einer sequentiellen Datei auf Diskette zu sichern. Das geht sehr leicht:

```
OPEN 2,8,2,"SPRITE,S,W":PRINT#2,MT$:CLOSE 2
```

schreibt die Sprite-Daten in die Datei SPRITE. Natürlich können Sie in einer Datei auch die Daten für mehrere Sprites hintereinander ablegen. Um vorhandene Sprite-Entwürfe verwenden zu können, gibt es die zweite Aufrufvariante des Editors:

```
GOSUB 1100
```

erwartet die darzustellenden Sprite-Daten in der Variablen MT$. Nachdem der Editor gestartet wurde und er das Sprite-Entwurfsraster, auch "Sprite-Matrix" genannt, auf den Bildschirm gezeichnet hat, blinkt der Cursor in der linken oberen Ecke des Bildschirms. Nun stehen Ihnen die folgenden Funktionen zur Verfügung:

<Cursor right>-Taste

Bewegt den Cursor um eine Position nach rechts.

<Cursor left>-Taste

Bewegt den Cursor um eine Position nach links.

<Cursor down>-Taste

Bewegt den Cursor um eine Position nach unten.

<Cursor up>-Taste

Bewegt den Cursor um eine Position nach oben.

<F1>-Taste

Setzt an der aktuellen Cursor-Position einen Punkt.

<F3>-Taste

Löscht den Punkt an der aktuellen Cursor-Position.

<F7>-Taste

Beendet die Eingabe und berechnet die Sprite-Daten.

<F8>-Taste

Beendet die Eingabe und bricht das Programm ab.

Die Cursor-Steuerung erkennt übrigens die Grenzen des Sprite-Rasters. Am Ende einer Zeile beispielsweise wechselt sie automatisch an den Beginn der nächsten Zeile. Die Cursor-Steuerung, die in den Programmzeilen 1.300 bis 1.520 realisiert ist, stellt sozusagen einen "verkleinerten" BASIC-Eingabe-Editor dar und läßt sich auch leicht für andere Zwecke, etwa für eine mehrzeilige Eingabe in ein Datenfeld einer Dateiverwaltung, einsetzen. Das aber nur als kleiner Programmiertip am Rande.

```
100 rem *****************************************************
105 rem *                                                   *
110 rem * programm: sprite-edit                             *
115 rem * editor zum entwerfen von sprites                  *
120 rem *                                                   *
125 rem *****************************************************
130 :
135 :
140 rem *** maschinenprogramm nachladen *******************
145 if fl=0 then fl=1:load "cursor",8,1
150 :
```

```
155 rem *** vorhandene matrix darstellen und veraendern?  **
160 rem *** ---> gosub 1100 (matrix in mt$)               **
165 rem ***                                               **
170 rem *** neue matrix definieren?                       **
175 rem *** ---> gosub 1250 (matrix befindet sich         **
180 rem ***                   anschliessend in mt$)       **
190 :
200 rem *** gewuenschten aufruf hier einfuegen! ************
210 gosub .....
220 :
230 :
1060 rem *** matrix darstellen *****************************
1100 print chr$(147)
1105 sw=peek(648)*256:rem home-position=matrixanfang
1110 for z1=0 to 60 step 3
1115 for z2=0 to 2
1120 aw=asc(mid$(mt$,z1+z2+1,1))
1125 for z3=0 to 7
1130 aw=aw/2
1140 if aw=int(aw) then pw=46:goto 1150
1145 pw=160
1150 poke sw+z1/3*40+z2*8+7-z3,pw:aw=int(aw)
1160 next z3
1170 next z2
1180 next z1
1190 goto 1300:rem zur matrixeingabesteuerung
1200 :
1220 rem *** matrix definieren *****************************
1250 print chr$(147):rem rasterfeld zeichnen
1255 sw=peek(648)*256
1260 for z1=0 to 20
1270 hv=sw+z1*40:for z2=0 to 23:poke hv+z2,46:next z2
1280 next z1
1290 :
1295 rem *** eingabesteuerung *******************************
1300 h=sw-peek(648)*256:rem startposition
1310 pz=int(h/40)+1:ps=h-(pz-1)*40+1
1320 ze=pz:sp=ps:rem cursor in startposition
1330 sys 49934,ze,sp:sys 49974:rem cursor setzen und ein
1335 poke 198,0:wait 198,1:get eg$:sys 49979: rem cursor aus
1340 eg=asc(eg$):rem aktuelle eingabe
1350 if not(eg=29) then 1380:rem cursor right
1360 if not(sp=ps+23) then sp=sp+1:goto 1330
1370 if not(ze=pz+20) then sp=ps:ze=ze+1:goto 1330
1380 if not(eg=157) then 1420:rem cursor left
1390 if not(sp=ps) then sp=sp-1:goto 1330
1400 if not(ze=pz) then sp=ps+23:ze=ze-1:goto 1330
1420 if not(eg=17) then 1440:rem cursor down
1430 if not(ze=pz+20) then ze=ze+1:goto 1330
1440 if not(eg=145) then 1460:rem cursor up
1450 if not(ze=pz) then ze=ze-1:goto 1330
1460 if not(eg=133) then 1480:rem f1=punkt setzen
1465 poke peek(648)*256+40*(ze-1)+sp-1,160
```

```
1470 if not(sp=ps+23) then sp=sp+1:goto 1330
1475 if not(ze=pz+20) then sp=ps:ze=ze+1:goto 1330
1480 if not(eg=134) then 1500:rem f3=punkt loeschen
1485 poke peek(648)*256+40*(ze-1)+sp-1,46
1490 if not(sp=ps+23) then sp=sp+1:goto 1330
1495 if not(ze=pz+20) then sp=ps:ze=ze+1:goto 1330
1500 if eg=136 then 1580:rem f7=matrix uebernehmen
1510 if eg=140 then end:rem f8=abbruch
1520 goto 1330
1550 :
1570 rem *** matrix uebernehmen ****************************
1580 sw=peek(648)*256:aw=0:mt$=""
1590 for z1=0 to 60 step 3
1600 for z2=0 to 2
1610 for z3=0 to 7
1620 pw=peek(sw+z1/3*40+z2*8+z3)
1630 if pw=160 then aw=aw+2^(7-z3)
1640 next z3
1650 mt$=mt$+chr$(aw):aw=0
1660 next z2
1670 next z1
1680 :
1690 return:rem fertig!
```

Alles abgetippt und abgespeichert? Gut, dann entwerfen Sie jetzt doch einmal mindestens zwei Sprites, damit Sie nachher etwas zum Experimentieren haben.

Vielleicht rätseln Sie im Augenblick noch etwas, wie Sie das Hauptprogramm des Editors gestalten sollen. Deshalb eine kleine Anregung:

```
210 print "neues sprite entwerfen : -1-"
220 print "altes sprite darstellen: -2-"
230 input "ihre wahl (1,2): ";wl
240 if wl<1 and wl>2 then 210:rem falsche eingabe
245 :
250 if wl=2 then 330
260 rem neues sprite
270 gosub 1250
280 print chr$(147)
290 input "name des sprites: ";sm$
300 open 2,8,2,sm$+",s,w":print#2,mt$:close 2
310 :
320 rem altes sprites
330 print chr$(147)
340 input "name des sprites: ";sm$
350 open 2,8,2,sm$+",s,r":input#2,mt$:close 2
360 gosub 1100
370 open 15,8,15,"s:"+sm$:close 15
```

```
380 open 2,8,2,sm$+",s,w":print#2,mt$:close 2
390 end
```

Das Programm fragt Sie, ob Sie ein neues Sprite entwerfen oder ein altes Sprite darstellen und verändern wollen. Ein neues Sprite wird anschließend unter dem erfragten Namen auf Diskette abgelegt.

Im zweiten Fall möchte das Programm wissen, unter welchem Namen das Sprite auf Diskette gespeichert ist. Zum Schluß wird es dann unter demselben Namen wieder auf die Diskette zurückgeschrieben. Natürlich können Sie das Hauptprogramm aber auch ganz anders gestalten. Aus diesem Grund habe ich es weiter oben bewußt weggelassen.

Zum Ausprobieren der nun folgenden Routinen sollten Sie, wie gesagt, über mindestens zwei Sprites verfügen, die Sie ganz nach Belieben entwerfen können. Am Ende des Abschnitts finden Sie wieder einen BASIC-Lader zu allen Maschinensprache-Routinen. Ein kleiner Tip: Wenn Sie den BASIC-Lader eingeben möchten, dann tun Sie das am besten jetzt gleich. Anschließend lesen Sie die folgenden Erklärungen durch und können dann am Rechner alles gleich praktisch ausprobieren. Das Assembler-Listing:

```
100 ;**********************************************************
105 ;*                                                        *
110 ;* programm: sprites                                      *
120 ;* routinen zur spriteprogrammierung                      *
140 ;*                                                        *
150 ;**********************************************************
200 ;
205 ;
210 .ba 51165   ;*** startadresse ***
220 ;
230 ;
240 ;*** labels ***********************************************
250 .gl zw  =  2   ;zwischenspeicher
260 .gl zp1 = 251
260 .gl zp2 = 252
260 .gl zp3 = 253
260 .gl zp4 = 254
270 ;
272 ;
274 ;**********************************************************
276 ;*                                                        *
278 ;* spdesign - aussehen eines sprites festlegen            *
```

```
280 ;*                                                          *
282 ;* aufruf: sys 51165,bl,da$                                 *
284 ;*      bl: blocknummer (speicherstartadresse/64)           *
286 ;*     da$: string mit sprite-daten (63 byte)               *
288 ;*                                                          *
290 ;************************************************************
300 ;
350 spdesign  jsr $aefd      ;auf komma testen
360           jsr blumw      ;blocknummer holen
370           jsr $aefd      ;auf komma testen
380           jsr $ad9e      ;datenstring holen
390           jsr $b6a3
400           cmp #63        ;laenge=63 byte?
410           beq spd1       ;ja, ok
420           jmp $b248      ;nein, fehler
430 spd1      jsr ramkon     ;auf ram umschalten
440           ldy #0
450 spd2      lda ($22),y    ;daten im speicher
460           sta ($14),y    ;ablegen
470           iny
480           cpy #63
490           bne spd2
500           jmp orgkon     ;auf rom zurueckschalten
```

SPDESIGN legt die in MT$ oder einer anderen String-Variablen gespeicherten Sprite-Daten in dem angegebenen Speicherbereich ab. Bitte beachten Sie, daß MT$ genau 63 Byte groß sein muß. Ansonsten erhalten Sie eine Fehlermeldung.

```
SYS 51165,13,MT$
```

legt die Daten Ihres einen Sprites im Speicher von Adresse 832 (13*64) bis 895 ab.

```
SYS 51165,14,MT$
```

legt die Daten des anderen Sprites von Adresse 896 bis 959 ab. Mit Hilfe der folgenden Routinen läßt sich die Sprite-Darstellung auch dann noch manipulieren, wenn Sie die Daten bereits im Speicher abgelegt haben:

```
510 ;
520 ;
522 ;************************************************************
524 ;*                                                          *
526 ;* spclr - spriteblock loeschen                             *
528 ;*                                                          *
530 ;* aufruf: sys 51204,bl                                     *
```

```
532 ;*     bl: blocknummer (speicherstartadresse/64)          *
534 ;*                                                        *
536 ;**********************************************************
540 ;
560 spclr      jsr $aefd        ;auf komma testen
570            jsr blumw        ;blocknummer holen
580            ldy #0           ;datenblock loeschen
590            tya
600 spc1       sta ($14),y
610            iny
620            cpy #63
630            bne spc1
640            rts              ;fertig!
650 ;
660 ;
662 ;**********************************************************
664 ;*                                                        *
666 ;* spinv - spriteblock invertieren                        *
668 ;*                                                        *
670 ;* aufruf: sys 51221,bl                                   *
672 ;*     bl: blocknummer (speicherstartadresse/64)          *
674 ;*                                                        *
676 ;**********************************************************
680 ;
700 spinv      jsr $aefd        ;auf komma testen
710            jsr blumw        ;blocknummer holen
720            jsr ramkon       ;auf ram umschalten
730            ldy #0           ;datenblock invertieren
740 spi1       lda ($14),y      ;byte holen
750            eor #$ff         ;komplementaerwert bilden
760            sta ($14),y
770            iny
780            cpy #63
790            bne spi1
800            jmp orgkon       ;auf rom zurueckschalten
810 ;
820 ;
822 ;**********************************************************
824 ;*                                                        *
826 ;* spmove - spriteblock verschieben                       *
828 ;*                                                        *
830 ;* aufruf: sys 51246,b1,b2                                *
832 ;*     b1: nummer des ausgangsblocks                      *
834 ;*     b2: nummer des zielblocks                          *
836 ;*                                                        *
838 ;**********************************************************
840 ;
860 spmove     jsr $aefd        ;auf komma testen
870            jsr gtbl2        ;blocknummern holen
880            lda zp3          ;zeiger umspeichern
890            ldy zp4
900            sta $22
910            sty $23
```

```
920            jmp spd1       ;zur spdesign-routine
930 ;
940 ;
942 ;*********************************************************
944 ;*                                                       *
946 ;* spchange - spriteblocks vertauschen                   *
948 ;*                                                       *
950 ;* aufruf: sys 51263,b1,b2                               *
952 ;*  b1,b2: nummern der zu vertauschenden blocks          *
954 ;*                                                       *
956 rem *****************************************************
960 ;
980 spchange   jsr $aefd      ;auf komma testen
990            jsr gtbl2      ;blocknummern holen
1000           jsr ramkon     ;auf ram umschalten
1010           ldy #0         ;datenblocks austauschen
1020 spch1     lda (zp3),y
1030           tax
1040           lda ($14),y
1050           sta (zp3),y
1060           txa
1070           sta ($14),y
1080           iny
1090           cpy #63
1100           bne spch1
1110           jmp orgkon     ;auf rom zurueckschalten
1120 ;
1130 ;
1132 ;********************************************************
1134 ;*                                                      *
1136 ;* spand - spriteblocks und-verknuepfen                 *
1138 ;*                                                      *
1140 ;* aufruf: sys 51292,b1,b2                              *
1142 ;*  b1,b2: nummern der zu verknuepfenden blocks         *
1144 ;*         (ergebnis kommt nach b1)                     *
1146 ;*                                                      *
1148 ;********************************************************
1150 ;
1170 spand     jsr $aefd      ;auf komma testen
1180           jsr gtbl2      ;blocknummern holen
1190           jsr ramkon     ;auf ram umschalten
1200           ldy #0         ;datenblocks und-verknuepfen
1210 spa1      lda (zp3),y
1220           and ($14),y
1230           sta (zp3),y
1240           iny
1250           cpy #63
1260           bne spa1
1270           jmp orgkon     ;auf rom zurueckschalten
1280 ;
1290 ;
1292 ;********************************************************
1294 ;*                                                      *
```

```
1296 ;* spor - spriteblocks oder-verknuepfen                  *
1298 ;*                                                       *
1300 ;* aufruf: sys 51317,b1,b2                               *
1302 ;*  b1,b2: nummern der zu verknuepfenden blocks          *
1304 ;*         (ergebnis kommt nach b1)                      *
1306 ;*                                                       *
1308 ;*********************************************************
1310 ;
1330 spor     jsr $aefd        ;auf komma testen
1340          jsr gtbl2        ;blocknummern holen
1350          jsr ramkon       ;auf ram umschalten
1360          ldy #0
1370 spo1     lda (zp3),y      ;datenblocks or-verknuepfen
1380          ora ($14),y
1390          sta (zp3),y
1400          iny
1410          cpy #63
1420          bne spo1
1430          jmp orgkon       ;auf rom zurueckschalten
1440 ;
1450 ;
1452 ;*********************************************************
1454 ;*                                                       *
1456 ;* speor - spriteblocks exclusiv-oder-verknuepfen        *
1458 ;*                                                       *
1460 ;* aufruf: sys 51342,b1,b2                               *
1462 ;*  b1,b2: nummern der zu verknuepfenden blocks          *
1464 ;*         (ergebnis kommt nach b1)                      *
1466 ;*                                                       *
1468 ;*********************************************************
1470 ;
1490 speor    jsr $aefd        ;auf komma testen
1500          jsr gtbl2        ;blocknummern holen
1510          jsr ramkon       ;auf ram umschalten
1520          ldy #0           ;datenblocks eor-verknuepfen
1530 spe1     lda (zp3),y
1540          eor ($14),y
1550          sta (zp3),y
1560          iny
1570          cpy #63
1580          bne spe1
1590          jmp orgkon       ;auf rom zurueckschalten
```

Mit Beispielen möchte ich noch etwas warten, da Sie sich diese ja im Moment noch nicht am Bildschirm anschauen können. Die folgende Routine SPBLOCK setzt die eingangs erwähnten Zeiger auf die mit SPDESIGN abgelegten Sprite-Daten.

```
1600 ;
1610 ;
1612 ;*******************************************************
1614 ;*                                                     *
1616 ;* spblock - ordnet sprite einen datenblock zu         *
1618 ;*                                                     *
1620 ;* aufruf: sys 51367,sc,nr,bl                          *
1622 ;*     sc: screennummer (speicherstartadresse/1024)    *
1624 ;*     nr: spritenummer (1-8)                          *
1626 ;*     bl: blocknummer (speicherstartadresse/64)       *
1628 ;*                                                     *
1630 ;*******************************************************
1640 ;
1650 spblock  jsr $aefd      ;auf komma testen
1660          jsr scrber     ;screennr. holen
1670          jsr $aefd      ;auf komma testen
1680          jsr gtsnr      ;spritenummer holen
1690          jsr $aefd      ;auf komma testen
1700          jsr gtblpar    ;blocknummer holen
1710          tya            ;low-byte in akku
1720          ldy zw         ;zeiger auf register
1730          sta (zp3),y
1740          rts            ;fertig!
```

Bei SPBLOCK taucht ein Parameter auf, den ich bisher noch nicht besprochen habe, die "Screen-Nummer". Damit ist die Nummer bzw. die Startadresse des Bildschirmspeichers gemeint. Wie ich Ihnen zu Beginn des Kapitels erklärt habe, liegt der Bildschirmspeicher normalerweise von Speicheradresse 1024 bis 2023. In diesem Fall müssen sie als Screen-Nummer eine 1 (1024/1024) angeben.

Die mit MBDESIGN in den Blöcken 13 und 14 abgelegten Daten Ihrer beiden Sprites weisen Sie mit

```
SYS 51367,1,1,13
SYS 51367,1,2,14
```

den Sprites 1 und 2 zu. Der Bildschirmspeicher läßt sich jedoch auch verschieben. Wenn Sie beispielsweise mit den Hires-Routinen aus dem letzten Abschnitt arbeiten und den Bitmap-Bereich 0 gewählt haben, befindet sich der Bildschirmspeicher der Adresse 52.224 bis 53.223. Als Screen-Nummer bei SPBLOCK müssen Sie deshalb eine 51 (52224/1024) angeben.

Außerdem müssen Sie die Sprite-Daten dann im Speicher irgendwo zwischen der Adresse 49.152 und 65.535 ablegen, da sich der Bildschirmspeicher und die Sprite-Daten immer in demselben 16-KByte-Speicherbereich befinden müssen. Um Kollisionen mit den Maschinensprache-Routinen aus den anderen und diesem Kapitel zu vermeiden, empfiehlt sich der Bereich direkt vor dem Bildschirmspeicher, beispielsweise

```
Block 812: 51968-52031
Block 813: 52032-52095
Block 814: 52096-52159
Block 815: 52160-52223
```

Mit den folgenden Routinen werden die "Sprite-Farben" festgelegt. Ähnlich wie bei der hochauflösenden Grafik haben Sie dabei die Auswahl zwischen zwei Farbmodi:

```
1750 ;
1760 ;
1762 ;*****************************************************
1764 ;*                                                   *
1766 ;* spmode - farbmodus festlegen                      *
1768 ;*                                                   *
1770 ;* aufruf: sys 51391,nr,md                           *
1772 ;*     nr: spritenummer (1-8)                        *
1774 ;*     md: 0 = einfarbmodus, 1 = mehrfarbmodus       *
1776 ;*                                                   *
1778 ;*****************************************************
1780 ;
1800 spmode   lda #<(53276)  ;register-Adresse
1810          ldy #>(53276)
1820          jmp flelsn     ;zum gemeinsamen teil
1830 ;
1840 ;
1842 ;*****************************************************
1844 ;*                                                   *
1846 ;* spcol - spritefarben festlegen                    *
1848 ;*                                                   *
1850 ;* aufruf: sys 51398,nr,fc                           *
1852 ;*     nr: spritenummer (1-8)                        *
1854 ;*     fc: farbcode (0-15)                           *
1856 ;*                                                   *
1858 ;*****************************************************
1860 ;
1890 spcol    jsr $aefd      ;auf komma testen
1900          jsr gtsnr      ;spritenummer holen
1910          jsr $aefd      ;auf komma testen
1920          jsr $b79e      ;farbcode holen
1930          txa
```

```
1940          ldy zw          ;zeiger auf register
1950          sta 53287,y     ;farbcode speichern
1960          rts             ;fertig!
1970 ;
1980 ;
1982 ;*****************************************************
1984 ;*                                                   *
1986 ;* spexcol - zusatzfarben fuer mehrfarbmodus         *
1988 ;*                                                   *
1990 ;* aufruf: sys 51417,f1,f2                           *
1992 ;*        f1: farbcode fuer bit-kombination '01'     *
1994 ;*        f2: farbcode fuer bit-kombination '11'     *
1996 ;*                                                   *
1998 ;*****************************************************
2000 ;
2020 spexcol  jsr $aefd       ;auf komma testen
2030          jsr $b79e       ;1. farbcode holen
2040          stx 53285       ;ins register schreiben
2050          jsr $aefd       ;auf komma testen
2060          jsr $b79e       ;2. farbcode holen
2070          stx 53286
2080          rts             ;fertig!
```

Der Einfarbmodus dürfte klar sein. Dort, wo Sie bei der Sprite-Definition einen Punkt gesetzt haben, erscheint dieser in der Farbe, die Sie mit SPCOL einstellen. An allen anderen Stellen scheint die Hintergrundfarbe durch. Im Mehrfarbmodus haben Sie die Auswahl zwischen vier Farben. Dazu werden jeweils zwei Bits der Sprite-Matrix zusammengefaßt. Die horizontale Auflösung der Sprites halbiert sich daher von 24 Punkten auf 12. Wenn man zwei Bits hat, ergeben sich vier Kombinationsmöglichkeiten: "00", "01", "10" und "11". Jeder dieser Bit-Kombinationen wird eine Farbe zugeordnet:

```
00: Hintergrundfarbe
01: Farbe F1 bei SPEXCOL
10: Mit SPCOL festgelegte Farbe
11: Farbe F2 bei SPEXCOL
```

Die Wahl zwischen Einfarb- und Mehrfarbmodus kann für jedes Sprite einzeln getroffen werden. Die beiden mit SPEXCOL festgelegten Zusatzfarben gelten jedoch für alle Mehrfarb-Sprites.

```
2090 ;
2100 ;
2102 ;*******************************************************
2104 ;*                                                     *
2106 ;* spprior - sprite-hintergrund-prioritaet             *
2108 ;*                                                     *
2110 ;* aufruf: sys 51436,nr,pr                             *
2112 ;*      nr: spritenummer (1-8)                         *
2114 ;*      pr: 0 = spriteprioritaet, 1 = hintergrundpr.   *
2116 ;*                                                     *
2118 ;*******************************************************
2120 ;
2140 spprior   lda #<(53275)   ;register-Adresse
2150           ldy #>(53275)
2160           jmp flelsn      ;zum gemeinsamen teil
```

Da die Sprites ja völlig unabhängig von der normalen Bildschirmdarstellung erzeugt werden und bis zu acht Sprites gleichzeitig dargestellt werden können, stellt sich bei Überlappungen zwischen mehreren Sprites einerseits und einem Sprite und dem Hintergrund (also dem Text- oder Grafikbildschirm) andererseits die Frage der Priorität. Soll beispielsweise ein Sprite vom Hintergrund verdeckt werden oder umgekehrt?

Bei den Sprites selbst sind die Prioritäten vordefiniert. Sprites mit höherer Nummer verdecken bei Überlappung Sprites mit niederer Nummer. Das Sprite 1 "verschwindet" also hinter allen anderen Sprites, das Sprite 2 nur hinter den Sprite 3 bis 8 usw...

Die Sprite-/Hintergrundpriorität läßt sich für jedes einzelne Sprite frei definieren. Eine 0 bei SPPRIOR bewirkt, daß das betreffende Sprite bei Überlappung den Hintergrund verdeckt. Eine 1 hat zur Folge, daß das Sprite hinter dem Hintergrund verschwindet.

```
2170 ;
2180 ;
2182 ;*******************************************************
2184 ;*                                                     *
2186 ;* spxsize - spritegroesse in x-richtung               *
2188 ;*                                                     *
2190 ;* aufruf: sys 51443,nr,md                             *
2192 ;*      nr: spritenummer (1-8)                         *
2194 ;*      md: 0 = normal, 1 = doppelte groesse           *
2196 ;*                                                     *
2198 ;*******************************************************
```

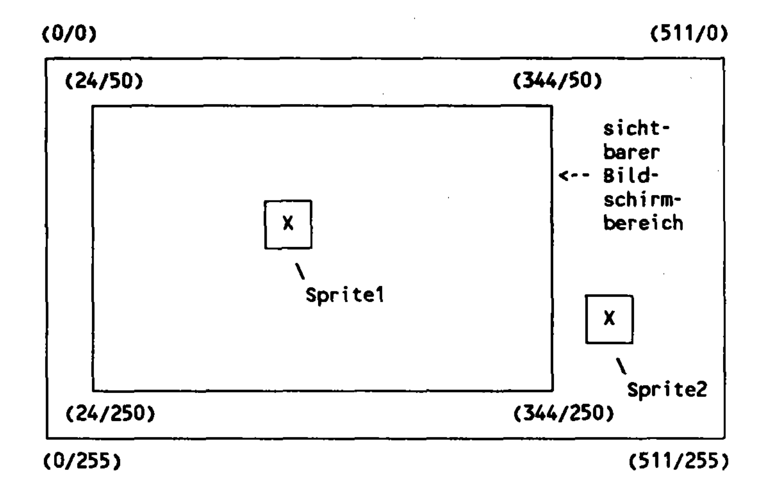
```
2200 ;
2230 spxsize  lda #<(53277)  ;register-adresse
2240          ldy #>(53277)
2250          jmp flelsn     ;zum gemeinsamen teil
2260 ;
2270 ;
2272 ;*****************************************************
2274 ;*                                                   *
2276 ;* spysize - spritegroesse in y-richtung             *
2278 ;*                                                   *
2280 ;* aufruf: sys 51450,nr,md                           *
2282 ;*     nr: spritenummer (1-8)                        *
2284 ;*     md: 0 = normal, 1 = doppelte groesse          *
2286 ;*                                                   *
2288 ;*****************************************************
2290 ;
2310 spysize  lda #<(53271)  ;register-adresse
2320          ldy #>(53271)
2330          jmp flelsn     ;zum gemeinsamen teil
```

Jedes Sprite kann in vier verschiedenen Größen dargestellt werden, die mit SPXSIZE und SPYSIZE eingestellt werden:

```
Horizontal (X): normal        Vertikal (Y): normal
Horizontal (X): doppelt groß  Vertikal (Y): normal
Horizontal (X): normal        Vertikal (Y): doppelt groß
Horizontal (X): doppelt groß  Vertikal (Y): doppelt groß
2340 ;
2350 ;
2352 ;*****************************************************
2354 ;*                                                   *
2356 ;* spsetpos - sprite positionieren                   *
2358 ;*                                                   *
2360 ;* aufruf: sys 51457,nr,x,y                          *
2362 ;*     nr: spritenummer (1-8)                        *
2364 ;*      x: x-koordinate (0-511)                      *
2366 ;*      y: y-koordinate (0-255)                      *
2368 ;*                                                   *
2370 ;*****************************************************
2380 ;
2390 spsetpos jsr $aefd      ;auf komma testen
2400          jsr gtsnr      ;spritenummer holen
2410          jsr $aefd      ;auf komma testen
2420          jsr $ad8a      ;x-koordinate holen
2430          jsr $b7f7      ;nach integer wandeln
2440          cmp #2         ;high-byte>1?
2450          bcc spsp1      ;nein, ok
2460          jmp $b248      ;ja, fehler
2470 spsp1    sty zp3
2480          sta zp4
2490          jsr $aefd      ;auf komma testen
2500          jsr $b79e      ;y-koordinate holen
```

```
2510        lda zw          ;spritenummer
2520        asl             ;*2
2530        tay
2540        txa             ;y-koord.
2550        sta 53249,y     ;in register ablegen
2560        lda zp3         ;x-koord. (low)
2570        sta 53248,y
2580        jsr rgbber      ;register-pos. fuer x-koord.
2590        ldx zp4         ;x-koord. (high)
2600        lda #<(53264)   ;register-adresse
2610        ldy #>(53264)
2620        sta zp3
2630        sty zp4
2640        jmp flelsn2     ;zum gemeinsamen teil
```

Wenn Sie sich erinnern, bei der hochauflösenden Grafik hatten wir eine Auflösung von 320 mal 200 Punkten. Das Koordinatensystem der Sprites nun reicht bis zur Position 511/255, geht also weit über den normalen Bildschirm hinaus. Ein Sprite kann sich daher durchaus an einer Position außerhalb des sichtbaren Bildschirms befinden. Die folgende Abbildung verdeutlicht diesen Sachverhalt:



Sprite 1 befindet sich etwa in der Mitte des Bildschirms. Es hat in etwa die Koordinaten 150/125. Sprite 2 dagegen befindet sich außerhalb des sichtbaren Bildschirmbereichs, etwa an der Position 375/200.

In der Abbildung sind die Sprite-Koordinaten der vier Eckpunkte des Bildschirms vermerkt. Der sichtbare Bereich der Sprites beginnt also bei der X-Koordinate 24 bzw. der Y-Koordinate 50 und endet bei X gleich 344 bzw. Y gleich 250.

Nun fragen Sie sich vielleicht, was das ganze überhaupt soll. Was soll man schließlich mit einem Sprite anfangen, das am Bildschirm gar nicht sichtbar ist? Die Frage ist durchaus berechtigt. Dadurch, daß das Sprite-Koordinatensystem größer als der normale Bildschirm ist, ergibt sich aber ein sehr interessanter Effekt: Man kann ein Sprite langsam bis zum völligen Verschwinden aus dem Bildschirm hinausbewegen lassen.

Bevor wir das ausprobieren können, benötigen Sie aber noch die folgenden beiden Routinen:

```
2650 ;
2660 ;
2662 ;*******************************************************
2664 ;*                                                     *
2666 ;* spon - sprite einschalten                           *
2668 ;*                                                     *
2670 ;* aufruf: sys 51518,nr                                *
2672 ;*     nr: spritenummer (1-8)                          *
2674 ;*                                                     *
2676 ;*******************************************************
2680 ;
2700 spon     lda #1          ;sprite ein
2710          .by $2c
2720 ;
2730 ;
2732 ;*******************************************************
2734 ;*                                                     *
2736 ;* spoff - sprite ausschalten                          *
2738 ;*                                                     *
2740 ;* aufruf: sys 51521,nr                                *
2742 ;*     nr: spritenummer (1-8)                          *
2744 ;*                                                     *
2746 ;*******************************************************
2750 ;
2770 spoff    lda #0          ;sprite aus
2780          sta zp2         ;flag zwischenspeichern
2790          jsr $aefd       ;auf komma testen
2800          jsr gtsnr       ;spritenummer holen
2810          jsr rgbber      ;register-pos. berechnen
2820          ldx zp2         ;flag
2830          lda #<(53269)   ;register-adresse
2840          ldy #>(53269)
```

```
2850         sta zp3
2860         sty zp4
2870         jmp flelsn2    ;zum gemeinsamen teil
```

Damit sind wir bei den beiden vielleicht wichtigsten Routinen angekommen. Sicher brennen Sie schon darauf, Ihre beiden Sprites endlich am Bildschirm betrachten zu können. Dazu geben sie einfach ein:

```
SYS 51518,1
SYS 51518,2
```

Insbesondere in Spielen ist es sehr wichtig, feststellen zu können, ob ein Sprite mit einem anderen Sprite oder dem Hintergrund zusammengestoßen ist. Der VIC-II-Chip verfügt dazu über zwei "Kollisions-Register":

```
2880 ;
2890 ;
2892 ;*****************************************************
2894 ;*                                                   *
2896 ;* spcoll - kollisions-register auslesen             *
2898 ;*                                                   *
2900 ;* aufruf: sys 51547                                 *
2902 ;*      peek(251): spritekollision                   *
2904 ;*      peek(252): sprite-hintergrund-kollision      *
2906 ;*                                                   *
2908 ;*****************************************************
2910 ;
2915 spcoll  ldy 53278      ;sprite-kollision
2920         ldx 53279      ;hintergrund-kollision
2930         sty zp1
2940         stx zp2
2950         rts            ;fertig
```

Mit SYS 51547 werden beide Kollisions-Register ausgelesen. Die Werte können Sie sich anschließend mit

```
SK=PEEK(251)
HK=PEEK(252)
```

in zwei Variablen holen. In beiden Registern ist je ein Bit für eines der acht Sprites zuständig. Kollidiert ein Sprite mit einem anderen Sprite oder dem Hintergrund, wird das entsprechende Bit des entsprechenden Registers gesetzt.

Dadurch ergibt sich ein bestimmter Register-Wert zwischen 0 und 255, der mit Hilfe der folgenden Tabelle leicht analysiert werden kann:

Sprite	1	2	3	4	5	6	7	8
Wert	1	2	4	8	16	32	64	128

Sind beispielsweise die Sprites 2 und 7 zusammengestoßen, enthält SK den Wert 2+64=66. Der Wert 145 (1+16+128) bedeutet, daß die Sprites 1, 5 und 8 miteinander kollidiert sind. Dasselbe gilt für die Sprite-Hintergrund-Kollision. Ist beispielsweise das Sprite 6 mit einer Hintergrunddarstellung zusammengestoßen, enthält HK den Wert 32.

Assembler-Programmierer vergessen nun bitte nicht, noch die folgenden gemeinsamen Unterprogramme abzutippen:

```
2960 ;
2970 ;
2980 ;
2990 ;
3000 ;
3040 ;*** gemeinsame unterprogramme ************************
3050 ;
3070 ;*** blocknummer holen und testen ***
3080 gtblpar  jsr $ad8a     ;blocknummer einlesen
3090          jsr $b7f7     ;und nach integer wandeln
3100          cmp #4        ;high-byte>3?
3110          bcc gtblp1    ;nein, ok
3120          jmp $b248     ;ja, fehler
3130 gtblp1   rts           ;fertig
3140 ;
3160 ;*** blocknummer holen, testen und umrechnen ***
3170 blumw    jsr gtblpar   ;blocknummer holen
3180          ldx #6        ;blocknummer*6
3190 blu1     asl $14       ;=absolute adresse
3200          rol $15
3210          dex
3220          bne blu1
3230          rts           ;fertig
3240 ;
3260 ;*** zwei blocknummern bearbeiten ***
3270 gtbl2    jsr blumw     ;1. blocknummer holen
3280          lda $14       ;und umspeichern
3290          ldy $15
3300          sta zp3
3310          sty zp4
```

```
3320          jsr $aefd      ;auf komma testen
3330          jmp blumw      ;2. blocknummer holen
3340 ;
3360 ;*** screen-nummer holen ***
3370 scrber   jsr $b79e      ;screen-nummer holen
3380          txa
3390          asl            ;screennr.*4
3400          asl            ;=absolute adresse
3410          clc            ;3*256 addieren
3420          adc #3
3430          sta zp4        ;als high-byte speichern
3440          lda #248       ;low-byte (zeigt auf 1. reg.)
3450          sta zp3
3460          rts            ;fertig
3470 ;
3490 ;*** register-bit entspr. spritenr. berechnen ***
3500 rgbber   lda #1
3510          ldy zw         ;spritenummer
3520 rgb1     beq rgbf       ;=0, fertig
3530          asl            ;bit um eine stelle nach links
3540          dey
3550          jmp rgb1
3560 rgbf     sta zp1        ;wert zwischenspeichern
3570          rts            ;fertig
3580 ;
3600 ;*** diverse flags einlesen und setzen ***
3610 flelsn   sta zp3        ;register-adresse setzen
3620          sty zp4
3630          jsr $aefd      ;auf komma testen
3640          jsr gtsnr      ;spritenummer holen
3650          jsr rgbber     ;register-position berechnen
3660          jsr $aefd      ;auf komma testen
3670          jsr $b79e      ;flag holen
3680 flelsn2  txa            ;=0?
3690          beq mbml
3700          cpx #1         ;=1?
3710          beq mbms
3720          jmp $b248      ;nein, fehler
3730 mbml     lda #255       ;and-maske berechnen
3740          sec
3750          sbc zp1
3760          ldy #0
3770          and (zp3),y    ;entspr. register-bit loeschen
3780          sta (zp3),y
3790          rts            ;fertig
3800 mbms     ldy #0
3810          lda zp1
3820          ora (zp3),y    ;entspr. register-bit setzen
3830          sta (zp3),y
3840          rts            ;fertig
3850 ;
3880 ;*** spritenummer holen ***
3890 gtsnr    jsr $b79e      ;spritenummer
```

```
3900          txa           ;=0?
3910          bne gts1      ;nein, ok
3920 gts2     jmp $b248     ;ja, fehler
3930 gts1     cpx #9        ;>8?
3940          bcs gts2      ;ja, fehler
3950          dex
3960          stx zw
3970          rts           ;fertig
3980 ;
4010 ;*** auf ram umschalten ***
4020 ramkon   sei           ;interrupt sperren
4030          lda #53
4040          sta $01
4050          rts           ;fertig
4060 ;
4090 ;*** auf rom umschalten ***
4100 orgkon   lda #55
4110          sta $01
4120          cli           ;interrupt freigeben
4130          rts           ;fertig
```

Und der BASIC-Lader:

```
100 rem ****************************************************
102 rem *                                                  *
104 rem * programm: sprites                                *
106 rem * programmlaenge: 546 bytes                        *
108 rem * routinen zur spriteprogrammierung                *
110 rem *                                                  *
112 rem ****************************************************
114 rem *                                                  *
116 rem * spdesign - aussehen eines sprites festlegen      *
118 rem *                                                  *
120 rem * aufruf: sys 51165,bl,da$                         *
122 rem *      bl: blocknummer (speicherstartadresse/64)   *
124 rem *     da$: string mit sprite-daten (63 byte)       *
126 rem *                                                  *
128 rem ****************************************************
130 rem *                                                  *
132 rem * spclr - spriteblock loeschen                     *
134 rem *                                                  *
136 rem * aufruf: sys 51204,bl                             *
138 rem *      bl: blocknummer (speicherstartadresse/64)   *
140 rem *                                                  *
142 rem ****************************************************
144 rem *                                                  *
146 rem * spinv - spriteblock invertieren                  *
148 rem *                                                  *
150 rem * aufruf: sys 51221,bl                             *
152 rem *      bl: blocknummer (speicherstartadresse/64)   *
154 rem *                                                  *
156 rem ****************************************************
```

```
158 rem *                                                  *
160 rem * spmove - spriteblock verschieben                 *
162 rem *                                                  *
164 rem * aufruf: sys 51246,b1,b2                          *
166 rem *     b1: nummer des ausgangsblocks                *
168 rem *     b2: nummer des zielblocks                    *
170 rem *                                                  *
172 rem ****************************************************
174 rem *                                                  *
176 rem * spchange - spriteblocks vertauschen              *
178 rem *                                                  *
180 rem * aufruf: sys 51263,b1,b2                          *
182 rem *  b1,b2: nummern der zu vertauschenden blocks     *
184 rem *                                                  *
186 rem ****************************************************
188 rem *                                                  *
190 rem * spand - spriteblocks und-verknuepfen             *
192 rem *                                                  *
194 rem * aufruf: sys 51292,b1,b2                          *
196 rem *  b1,b2: nummern der zu verknuepfenden blocks     *
198 rem *         (ergebnis kommt nach b1)                 *
200 rem *                                                  *
202 rem ****************************************************
204 rem *                                                  *
206 rem * spor - spriteblocks oder-verknuepfen             *
208 rem *                                                  *
210 rem * aufruf: sys 51317,b1,b2                          *
212 rem *  b1,b2: nummern der zu verknuepfenden blocks     *
214 rem *         (ergebnis kommt nach b1)                 *
216 rem *                                                  *
218 rem ****************************************************
220 rem *                                                  *
222 rem * speor - spriteblocks exclusiv-oder-verknuepfen   *
224 rem *                                                  *
226 rem * aufruf: sys 51342,b1,b2                          *
228 rem *  b1,b2: nummern der zu verknuepfenden blocks     *
230 rem *         (ergebnis kommt nach b1)                 *
232 rem *                                                  *
234 rem ****************************************************
236 rem *                                                  *
238 rem * spblock - ordnet sprite einen datenblock zu      *
240 rem *                                                  *
242 rem * aufruf: sys 51367,sc,nr,bl                       *
244 rem *     sc: screennummer (speicherstartadresse/1024) *
246 rem *     nr: spritenummer (1-8)                       *
248 rem *     bl: blocknummer (speicherstartadresse/64)    *
250 rem *                                                  *
252 rem ****************************************************
254 rem *                                                  *
256 rem * spmode - farbmodus festlegen                     *
258 rem *                                                  *
260 rem * aufruf: sys 51391,nr,md                          *
262 rem *     nr: spritenummer (1-8)                       *
```

```
264 rem *     md: 0 = einfarbmodus, 1 = mehrfarbmodus        *
266 rem *                                                    *
268 rem ******************************************************
270 rem *                                                    *
272 rem * spcol - spritefarben festlegen                     *
274 rem *                                                    *
276 rem * aufruf: sys 51398,nr,fc                            *
278 rem *     nr: spritenummer (1-8)                         *
280 rem *     fc: farbcode (0-15)                            *
282 rem *                                                    *
284 rem ******************************************************
286 rem *                                                    *
288 rem * spexcol - zusatzfarben fuer mehrfarbmodus          *
290 rem *                                                    *
292 rem * aufruf: sys 51417,f1,f2                            *
294 rem *     f1: farbcode fuer bit-kombination '01'         *
296 rem *     f2: farbcode fuer bit-kombination '11'         *
298 rem *                                                    *
300 rem ******************************************************
302 rem *                                                    *
304 rem * spprior - sprite-hintergrund-prioritaet            *
306 rem *                                                    *
308 rem * aufruf: sys 51436,nr,pr                            *
310 rem *     nr: spritenummer (1-8)                         *
312 rem *     pr: 0 = spriteprioritaet, 1 = hintergrundpr. *
314 rem *                                                    *
316 rem ******************************************************
318 rem *                                                    *
320 rem * spxsize - spritegroesse in x-richtung              *
322 rem *                                                    *
324 rem * aufruf: sys 51443,nr,md                            *
326 rem *     nr: spritenummer (1-8)                         *
328 rem *     md: 0 = normal, 1 = doppelte groesse           *
330 rem *                                                    *
332 rem ******************************************************
334 rem *                                                    *
336 rem * spysize - spritegroesse in y-richtung              *
338 rem *                                                    *
340 rem * aufruf: sys 51450,nr,md                            *
342 rem *     nr: spritenummer (1-8)                         *
344 rem *     md: 0 = normal, 1 = doppelte groesse           *
346 rem *                                                    *
348 rem ******************************************************
350 rem *                                                    *
352 rem * spsetpos - sprite positionieren                    *
354 rem *                                                    *
356 rem * aufruf: sys 51457,nr,x,y                           *
358 rem *     nr: spritenummer (1-8)                         *
360 rem *      x: x-koordinate (0-511)                       *
362 rem *      y: y-koordinate (0-255)                       *
364 rem *                                                    *
366 rem ******************************************************
368 rem *                                                    *
```

```
370 rem * spon - sprite einschalten                          *
372 rem *                                                    *
374 rem * aufruf: sys 51518,nr                               *
376 rem *     nr: spritenummer (1-8)                         *
378 rem *                                                    *
380 rem ******************************************************
382 rem *                                                    *
384 rem * spoff - sprite ausschalten                         *
386 rem *                                                    *
388 rem * aufruf: sys 51521,nr                               *
390 rem *     nr: spritenummer (1-8)                         *
392 rem *                                                    *
394 rem ******************************************************
396 rem *                                                    *
398 rem * spcoll - kollisions-register auslesen              *
400 rem *                                                    *
402 rem * aufruf: sys 51547                                  *
404 rem *     peek(251): spritekollision                     *
406 rem *     peek(252): sprite-hintergrund-kollision        *
408 rem *                                                    *
410 rem ******************************************************
412 :
414 :
415 rem *** datas einlesen ***
420 restore:zl=1000
425 ad=51165:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 32,253,174,32,116,201,32,253,174,32,1299
1010 data 158,173,32,163,182,201,63,240,3,76,1291
1020 data 72,178,32,243,201,160,0,177,34,145,1242
1030 data 20,200,192,63,208,247,76,249,201,32,1488
1040 data 253,174,32,116,201,160,0,152,145,20,1253
1050 data 200,192,63,208,249,96,32,253,174,32,1499
1060 data 116,201,32,243,201,160,0,177,20,73,1223
1070 data 255,145,20,200,192,63,208,245,76,249,1653
1080 data 201,32,253,174,32,129,201,165,253,164,1604
1090 data 254,133,34,132,35,76,243,199,32,253,1391
1100 data 174,32,129,201,32,243,201,160,0,177,1349
1110 data 253,170,177,20,145,253,138,145,20,200,1521
1120 data 192,63,208,241,76,249,201,32,253,174,1689
1130 data 32,129,201,32,243,201,160,0,177,253,1428
1140 data 49,20,145,253,200,192,63,208,245,76,1451
1150 data 249,201,32,253,174,32,129,201,32,243,1546
```

```
1160 data 201,160,0,177,253,17,20,145,253,200,1426
1170 data 192,63,208,245,76,249,201,32,253,174,1693
1180 data 32,129,201,32,243,201,160,0,177,253,1428
1190 data 81,20,145,253,200,192,63,208,245,76,1483
1200 data 249,201,32,253,174,32,146,201,32,253,1573
1210 data 174,32,226,201,32,253,174,32,102,201,1427
1220 data 152,164,2,145,253,96,169,28,160,208,1377
1230 data 76,176,201,32,253,174,32,226,201,32,1403
1240 data 253,174,32,158,183,138,164,2,153,39,1296
1250 data 208,96,32,253,174,32,158,183,142,37,1315
1260 data 208,32,253,174,32,158,183,142,38,208,1428
1270 data 96,169,27,160,208,76,176,201,169,29,1311
1280 data 160,208,76,176,201,169,23,160,208,76,1457
1290 data 176,201,32,253,174,32,226,201,32,253,1580
1300 data 174,32,138,173,32,247,183,201,2,144,1326
1310 data 3,76,72,178,132,253,133,254,32,253,1386
1320 data 174,32,158,183,165,2,10,168,138,153,1183
1330 data 1,208,165,253,153,0,208,32,162,201,1383
1340 data 166,254,169,16,160,208,133,253,132,254,1745
1350 data 76,195,201,169,1,44,169,0,133,252,1240
1360 data 32,253,174,32,226,201,32,162,201,166,1479
1370 data 252,169,21,160,208,133,253,132,254,76,1658
1380 data 195,201,172,30,208,174,31,208,132,251,1602
1390 data 134,252,96,32,138,173,32,247,183,201,1488
1400 data 4,144,3,76,72,178,96,32,102,201,908
1410 data 162,6,6,20,38,21,202,208,249,96,1008
1420 data 32,116,201,165,20,164,21,133,253,132,1237
1430 data 254,32,253,174,76,116,201,32,158,183,1479
1440 data 138,10,10,24,105,3,133,254,169,248,1094
1450 data 133,253,96,169,1,164,2,240,5,10,1073
1460 data 136,76,166,201,133,251,96,133,253,132,1577
1470 data 254,32,253,174,32,226,201,32,162,201,1567
1480 data 32,253,174,32,158,183,138,240,7,224,1441
1490 data 1,240,15,76,72,178,169,255,56,229,1291
1500 data 251,160,0,49,253,145,253,96,160,0,1367
1510 data 165,251,17,253,145,253,96,32,158,183,1553
1520 data 138,208,3,76,72,178,224,9,176,249,1333
1530 data 202,134,2,96,120,169,53,133,1,96,1006
1540 data 169,55,133,1,88,96,0,0,0,0,542
2000 data -1:rem endmarkierung
```

Nachdem wir nun alle Routinen beieinander haben, können wir uns einmal ein größeres Beispiel im Zusammenhang ansehen. Noch einmal kurz zur Wiederholung:

Ihre beiden Sprites, die Sie sich mit dem Sprite-Editor definiert haben, haben Sie mit:

```
100 open 2,8,2,"sprite1,s,r":input mt$:close 2
110 sys 51165,13,mt$:rem sprite-daten in speicher
```

```
120 open 2,8,2,"sprite2,s,r":input mt$:close 2
130 sys 51165,14,mt$:rem sprite-daten in speicher
```

im Speicher abgelegt und mit:

```
140 sys 51367,1,1,13:rem zeiger von sprite 1 auf block 13
150 sys 51367,1,2,14:rem zeiger von sprite 2 auf block 14
```

den Sprites 1 und 2 zugewiesen. Eingeschaltet werden die beiden Sprites nun durch

```
160 sys 51518,1:rem sprite 1 ein
170 sys 51518,2:rem sprite 2 ein
```

Vielleicht möchten Sie den Sprites eine bestimmte Farbe geben. 16 Farben stehen zur Auswahl:

```
0: schwarz    8: orange
1: weiß       9: braun
2: rot       10: hellrot
3: türkis    11: grau 1
4: violett   12: grau 2
5: grün      13: hellgrün
6: blau      14: hellblau
7: gelb      15: grau 3
```

Gesetzt werden die Farben mit

```
180 sys 51398,1,5:rem farbe fuer sprite 1
190 sys 51398,2,6:rem farbe fuer sprite 2
```

Schließlich müssen wir die Sprites noch an eine bestimmte Bildschirmposition setzen. Im Moment befinden sie sich höchstwahrscheinlich an den Koordinaten 0/0 und sind daher nicht sichtbar.

```
200 sys 51457,100,100:rem koordinaten sprite 1
210 sys 51457,100,150:rem koordinaten sprite 2
```

Wenn Sie das Programm jetzt starten, werden die beiden Sprites knapp untereinander in die linke Bildschirmhälfte gesetzt. Mit

```
SYS 51443,1,1:SYS 51450,1,1
```

bringen Sie Sprite 1 auf die doppelte Größe. Probieren Sie auch einmal die verschiedenen Verknüpfungsroutinen aus.

```
SYS 51342,13,14
```

beispielsweise verknüpft die beiden Sprite-Matrizen logisch EXCLUSIV-ODER. Um die Sprites zum Schluß wieder auszuschalten, geben Sie ein:

```
SYS 51521,1:sys 51521,2
```

Am interessantesten ist es, ein Sprite in "fließende" Bewegung zu versetzen. Mit Hilfe der Routine MBSETPOS und einer schlichten FOR-NEXT-Schleife läßt sich das sehr leicht erreichen:

```
100 open 2,8,2,"sprite1,s,r":input mt$:close 2
110 sys 51165,13,mt$:rem sprite-daten in speicher
120 sys 51367,1,1,13:rem zeiger von sprite 1 auf block 13
130 sys 51518,1:rem sprite 1 ein
140 :
150 for x=0 to 511 step 1
160 sys 51457,x,150
170 next x
180 :
190 sys 51521,1:rem sprite 1 aus
```

Wenn Sie das Programm starten, wandert das Sprite langsam horizontal über den Bildschirm. Verändern Sie nun einmal die Schrittweite in Programmzeile 150. Bei einer Schrittweite ab etwa 5 huscht das Sprite schon blitzschnell über den Bildschirm. Möchte man die Flugbahn völlig frei definieren können, legt man die verschiedenen Koordinaten am besten in DATA-Zeilen ab:

```
100 open 2,8,2,"sprite1,s,r":input mt$:close 2
110 sys 51165,13,mt$:rem sprite-daten in speicher
120 sys 51367,1,1,13:rem zeiger von sprite 1 auf block 13
130 sys 51518,1:rem sprite 1 ein
140 :
150 read x,y,vz
160 if x=-1 then 210:rem fertig
170 sys 51457,x,y
180 for w=1 to vz:next w:rem verzoegerungsschleife
190 goto 150
200 :
210 sys 51521,1:rem sprite 1 aus
```

```
220 :
230 :
240 rem koordinatenpaare (x/y) und verzoegerungswert
250 data 24,50,200
260 data 344,250,200
270 data 344,50,200
280 data 24,250,200
290 data -1,-1,-1
```

Die DATA-Werte setzen Sprite 1 nacheinander in die vier Ecken des Bildschirms. Zugegeben, keine besonders originelle "Flugbahn".

Ihnen sind in der Zwischenzeit aber sicher schon die tollsten Ideen gekommen. Probieren Sie sie aus!

6.4 Zeichensätze

In diesem Abschnitt möchte ich Ihnen zeigen, wie man sich auf dem Commodore 64 eigene Zeichen definieren kann.

Drei Zeichen, die wohl schon jeder einmal schmerzlich vermißt hat, sind die deutschen Umlaute "ä", "ö" und "ü".

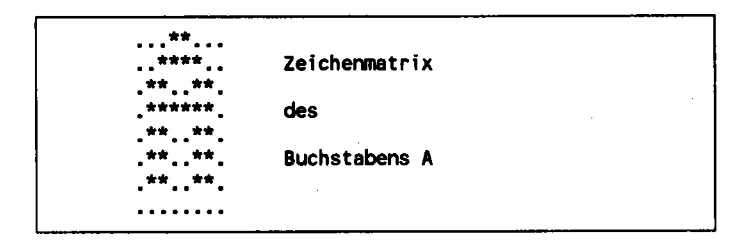
Am Ende des Abschnitts stelle ich Ihnen deshalb ein kleines Programm vor, das auf dem Commdore 64 einen deutschen Zeichensatz einrichtet.

6.4.1 Grundlagen

Alle auf dem Commodore 64 darstellbaren Zeichen sind in einem Zeichensatz zusammengefaßt, der im sogenannten Zeichengenerator oder Zeichensatz-ROM abgelegt ist.

Im Zeichengenerator ist das Aussehen jedes Zeichens in codierter Form gespeichert.

Jedes Zeichen ist aus 8 mal 8 Punkten aufgebaut, man spricht auch von einer 8x8-Matrix, das "A" z.B. so:



Zur Codierung einer Zeichenmatrix benötigt man acht Bytes, für eine Matrixzeile, die ja gerade aus acht Punkten besteht, jeweils eines. Jedes Bit eines Bytes ist dabei für einen Matrixpunkt zuständig.

Ist der betreffende Matrixpunkt gesetzt, hat das Bit den Wert 1, ansonsten den Wert 0. Die erste Matrixzeile des Buchstabens A beispielsweise wird durch den Wert 24 (16+8) codiert.

Der gesamte Zeichensatz besteht aus 512 Zeichen. 512 mal 8, das sind 4.096 Bytes. Der Zeichengenerator belegt also 4.096 Bytes im Speicher.

Wie Sie ja wissen, kann man durch gleichzeitiges Drücken der <Shift>- und der <Commodore>-Taste zwischen zwei Zeichendarstellungen umschalten. Entsprechend ist auch der Zeichensatz in zwei Teilsätze zu je 256 Zeichen aufgeteilt.

Die folgende Tabelle gibt Ihnen einen Überblick über die "Lage" aller Zeichen innerhalb des Zeichensatzes. Der Einfachheit halber sind nur die ASCII- und die Bildschirmcodes der Zeichen angegeben.

Die zu den Codes gehörenden Zeichen können Sie aus den beiden Tabellen im Anhang ersehen:

		Teilzeichensatz 1	
Zeichennr.	**POKE-Code**	**ASCII-Code**	**Zeichen**
		Normalschrift	
0 - 31	0 - 31	64 - 95	@ bis <- und Großbuchstaben
32 - 63	32 - 63	32 - 63	SPACE bis ?
64 - 95	64 - 95	96 - 127	Grafikzeichen
96 - 127	96 - 127	160 - 191	Grafikzeichen
		Reversschrift	
128 - 159	128 - 159	(18+) 64 - 95	@ bis <- und Großbuchstaben
160 - 191	160 - 191	(18+) 32 - 63	SPACE bis ?
192 - 223	192 - 223	(18+) 96 - 127	Grafikzeichen
224 - 255	224 - 255	(18+) 160 - 191	Grafikzeichen
		Teilzeichensatz 2	
Zeichennr.	**POKE-Code**	**ASCII-Code**	**Zeichen**
		Normalschrift	
256 - 287	0 - 31	64 - 95	@ bis <- und Kleinbuchstaben
288 - 319	32 - 63	32 - 63	SPACE bis ? und
320 - 351	64 - 95	96 - 127	Großbuchstaben Grafikzeichen
352 - 383	96 - 127	160 - 191	Grafikzeichen
		Reversschrift	
384 - 415	128 - 159	(18+) 64 - 95	@ bis <- und Kleinbuchstaben
416 - 447	160 - 191	(18+) 32 - 63	SPACE bis ? und
448 - 479	192 - 223	(18+) 96 - 127	Großbuchstaben Grafikzeichen
480 - 511	224 - 255	(18+) 160 - 191	Grafikzeichen

Der Zeichengenerator befindet sich normalerweise von Speicheradresse 53.248 bis 57.343. Er läßt sich aber von dort auslesen und in einen anderen Speicherbereich verschieben. Dort können wir die Codes der einzelnen Zeichen und damit das Aussehen der Zeichen ganz nach Belieben ändern! Zunächst stellt sich aber die Frage, wie man den Zeichengenerator ausliest

und wo man ihn am besten im Speicher ablegt. Schließlich benötigt er ja immerhin 4.096 Bytes an Speicherplatz.

Das Problem des Auslesens ist schnell gelöst. Dazu nehmen wir einfach die TRANSFER-Routine zum Verschieben von Speicherbereichen aus Kapitel 4. An Speicherbereichen stehen im Prinzip sieben zur Auswahl:

Nummer	Speicherbereich
0	53248 - 57343
1	0 - 4095
2	4096 - 8191
3	8192 - 12287
4	12288 - 16383
5	57344 - 61439
6	61440 - 65535

Im Bereich 0 liegt das Zeichensatz-ROM, das nur gelesen werden kann. Der Bereich 1 ist nur sehr bedingt zu empfehlen, da das Betriebssystem des Commodore 64 in diesem Bereich zahlreiche wichtige Werte ablegt, die durch den Zeichensatz überschrieben würden.

Als am günstigsten erweist sich der Bereich 4 von Speicheradresse 12.288 bis 16.383. In diesem Speicherbereich werden wir auch im folgenden experimentieren.

6.4.2 Eigene Zeichensätze programmieren

Das wichtigste Hilfsmittel zur Definition eigener Zeichen ist ein Zeichen-Editor, mit dem Sie die Zeichenmatrix erstellen können. Der im folgenden abgedruckte Zeichen-Editor ist in BASIC geschrieben. Um Ihnen aber ein möglichst komfortables Editieren der Zeichen zu ermöglichen, arbeitet er mit einer kleinen Maschinensprache-Routine, die den Cursor an eine vorgegebene Bildschirmposition setzt.

Die Routine CURSOR habe ich Ihnen bereits im letzten Abschnitt vorgestellt. Falls Sie das Programm noch nicht abgetippt

haben, holen Sie das bitte jetzt nach. Der Zeichen-Editor ist ohne diese Routine nicht lauffähig.

Die Bedienung des Zeichen-Editors gestaltet sich sehr einfach. Der Editor ist als Unterprogramm geschrieben, wobei es zwei Aufrufvarianten gibt: Möchten Sie ein Zeichen ganz neu entwerfen, dann rufen Sie den Editor mit

```
GOSUB 1250
```

auf. Die berechneten Zeichenmatrix-Daten befinden sich anschließend in der Variablen MT$. Um ein vorhandenes Zeichen abändern zu können, gibt es die zweite Aufrufvariante:

```
GOSUB 1100
```

erwartet die darzustellenden Zeichenmatrix-Daten in der Variablen MT$. Nachdem der Editor gestartet wurde und er das Entwurfsraster auf den Bildschirm gezeichnet hat, blinkt der Cursor in der linken oberen Ecke des Bildschirms.

Nun stehen Ihnen die folgenden Funktionen zur Verfügung:

<Cursor right>-Taste

Bewegt den Cursor um eine Position nach rechts.

<Cursor left>-Taste

Bewegt den Cursor um eine Position nach links.

<Cursor down>-Taste

Bewegt den Cursor um eine Position nach unten.

<Cursor up>-Taste

Bewegt den Cursor um eine Position nach oben.

<F1>-Taste

Setzt an der aktuellen Cursor-Position einen Punkt.

<F3>-Taste

Löscht den Punkt an der aktuellen Cursor-Position.

<F7>-Taste

Beendet die Eingabe und berechnet die Zeichendaten.

<F8>-Taste

Beendet die Eingabe und bricht das Programm ab.

Bitte geben Sie nun das Programm ein:

```
100 rem ******************************************************
105 rem *                                                    *
110 rem * programm: char-edit                                *
115 rem * editor zum entwerfen von zeichen                   *
120 rem *                                                    *
125 rem ******************************************************
130 :
135 :
140 rem *** maschinenprogramm nachladen *******************
145 if fl=0 then fl=1:load "cursor",8,1
150 :
155 rem *** vorhandene matrix darstellen und veraendern?  **
160 rem *** ---> gosub 1100 (matrix in mt$)              **
165 rem ***                                               **
170 rem *** neue matrix definieren?                       **
175 rem *** ---> gosub 1250 (matrix befindet sich         **
180 rem ***                   anschliessend in mt$)       **
190 :
200 rem *** gewuenschten aufruf hier einfuegen! ************
210 gosub .....
220 :
230 :
1060 rem *** matrix darstellen ****************************
1100 print chr$(147)
1105 sw=peek(648)*256:rem home-position=matrixanfang
1110 for z1=0 to 7
1120 aw=asc(mid$(mt$,z1+1,1))
1125 for z2=0 to 7
1130 aw=aw/2
1140 if aw=int(aw) then pw=46:goto 1150
```

```
1145 pw=160
1150 poke sw+z1*40+7-z2,pw:aw=int(aw)
1160 next z2
1180 next z1
1190 goto 1300:rem zur matrixeingabesteuerung
1200 :
1220 rem *** matrix definieren ******************************
1250 print chr$(147):rem rasterfeld zeichnen
1255 sw=peek(648)*256
1260 for z1=0 to 7
1270 hv=sw+z1*40:for z2=0 to 7:poke hv+z2,46:next z2
1280 next z1
1290 :
1295 rem *** eingabesteuerung ******************************
1300 h=sw-peek(648)*256:rem startposition
1310 pz=int(h/40)+1:ps=h-(pz-1)*40+1
1320 ze=pz:sp=ps:rem cursor in startposition
1330 sys 49934,ze,sp:sys 49974:rem cursor setzen und ein
1335 poke 198,0:wait 198,1:get eg$:sys 49979: rem cursor aus
1340 eg=asc(eg$):rem aktuelle eingabe
1350 if not(eg=29) then 1380:rem cursor right
1360 if not(sp=ps+7) then sp=sp+1:goto 1330
1370 if not(ze=pz+7) then sp=ps:ze=ze+1:goto 1330
1380 if not(eg=157) then 1420:rem cursor left
1390 if not(sp=ps) then sp=sp-1:goto 1330
1400 if not(ze=pz) then sp=ps+7:ze=ze-1:goto 1330
1420 if not(eg=17) then 1440:rem cursor down
1430 if not(ze=pz+7) then ze=ze+1:goto 1330
1440 if not(eg=145) then 1460:rem cursor up
1450 if not(ze=pz) then ze=ze-1:goto 1330
1460 if not(eg=133) then 1480:rem f1=punkt setzen
1465 poke peek(648)*256+40*(ze-1)+sp-1,160
1470 if not(sp=ps+7) then sp=sp+1:goto 1330
1475 if not(ze=pz+7) then sp=ps:ze=ze+1:goto 1330
1480 if not(eg=134) then 1500:rem f3=punkt loeschen
1485 poke peek(648)*256+40*(ze-1)+sp-1,46
1490 if not(sp=ps+7) then sp=sp+1:goto 1330
1495 if not(ze=pz+7) then sp=ps:ze=ze+1:goto 1330
1500 if eg=136 then 1580:rem f7=matrix uebernehmen
1510 if eg=140 then end:rem f8=abbruch
1520 goto 1330
1550 :
1570 rem *** matrix uebernehmen ****************************
1580 sw=peek(648)*256:aw=0:mt$=""
1590 for z1=0 to 7
1600 for z2=0 to 7
1620 pw=peek(sw+z1*40+z2)
1630 if pw=160 then aw=aw+2^(7-z2)
1640 next z2
1650 mt$=mt$+chr$(aw):aw=0
1670 next z1
1680 :
1690 return:rem fertig!
```

Das Hauptprogramm des Editors habe ich noch bewußt weggelassen. Wir werden es weiter unten ergänzen. Zunächst möchte ich Ihnen wieder eine kleine Routinensammlung vorstellen, die die Zeichensatzprogrammierung erheblich vereinfacht:

```
100 ;*********************************************************
105 ;*                                                       *
110 ;* programm: zeichensatz                                 *
120 ;* routinen zur zeichensatzprogrammierung                *
140 ;*                                                       *
150 ;*********************************************************
200 ;
205 ;
210 .ba 51711   ;*** startadresse ***
220 ;
230 ;
240 ;*** labels ********************************************
245 .gl zw  =  02   ;zwischenspeicher
250 .gl zp1 = 251
255 .gl zp2 = 252
260 .gl zp3 = 253
265 .gl zp4 = 254
270 ;
280 ;
282 ;*********************************************************
284 ;*                                                       *
286 ;* charset - aktuellen zeichensatz festlegen             *
288 ;*                                                       *
290 ;* aufruf: sys 51711,nr                                  *
292 ;*     nr: nummer des zeichensatzes                      *
294 ;*         0: 53248-57343 (original!)                    *
296 ;*         1:     0- 4095                                *
298 ;*         2   4096- 8191                                *
300 ;*         3   8192-12287                                *
302 ;*         4  12288-16383                                *
304 ;*         5  57344-61439                                *
306 ;*         6  61440-65535                                *
308 ;*                                                       *
310 ;*********************************************************
320 ;
340 charset    jsr $aefd      ;auf komma testen
350            jsr gtchsnr    ;zeichensatznummer
360            stx chaklfl    ;in flag ablegen
370            txa            ;und in akku
380            asl            ;*2
390            tay
400            lda chrtab1,y  ;basisadresse
410            sta chadrl
420            lda chrtab1+1,y
430            sta chadrh
440            rts            ;fertig
```

Als erstes müssen Sie festlegen, in welchem Speicherbereich der Zeichensatz liegen soll. Da Sie den Zeichensatz kaum komplett neu erstellen wollen, lesen Sie mit Hilfe der TRANSFER-Routine aus Kapitel 4 den Originalzeichensatz aus. Dazu laden Sie die TRANSFER-Routine und geben dann ein:

```
SYS 49152,53248,57343,12288,1
```

TRANSFER kopiert den Zeichensatz in Sekundenschnelle in den Bereich 4 ab Speicheradresse 12.288. Anschließend teilen Sie den Zeichensatzroutinen mit

```
SYS 51711,4
```

mit, daß Sie auf diesen Bereich wirken sollen. Die folgenden Routinen bilden den zentralen Bestandteil des ganzen Pakets:

```
450 ;
452 ;
454 ;**********************************************************
456 ;*                                                        *
458 ;* chread - matrix-byte eines zeichens auslesen           *
460 ;*                                                        *
462 ;* aufruf: sys 51736,zn,mp                                *
464 ;*     zn: zeichencode (0-511)                            *
466 ;*     mp: matrixposition (1-8)                           *
468 ;*                                                        *
470 ;**********************************************************
475 ;
480 chread     jsr $aefd      ;auf komma testen
490            jsr gtchrcd    ;zeichencode holen
500            jsr $aefd      ;auf komma testen
510            jst gtsnr      ;position innerhalb matrix
520            sei            ;interrupt sperren
530            lda #53        ;ram
540            ldx chaklfl    ;aktueller zeichensatz
550            bne chrd1
560            lda #51        ;original-zeichensatz
570 chrd1      sta $01        ;umschalten
580            ldy zw         ;position innerhalb matrix
590            lda ($14),y    ;byte holen
600            sta zw
610            jsr orgkon     ;auf rom zurueckschalten
620            ldy zw
630            sty zp1        ;matrix-byte
640            rts            ;fertig
650 ;
652 ;
```

```
654 ;*********************************************************
656 ;*                                                       *
658 ;* chwrite - matrix-byte eines zeichens schreiben        *
660 ;*                                                       *
662 ;* aufruf: sys 51774,zn,mp,wt                            *
664 ;*      zn: zeichencode (0-511)                          *
666 ;*      mp: matrixposition (1-8)                         *
668 ;*      wt: matrixcode (0-255)                           *
670 ;*                                                       *
672 ;*********************************************************
675 ;
680 chwrite    jsr chtest      ;zeichensatz testen
690            jsr $aefd       ;auf komma testen
700            jsr gtchrcd     ;zeichencode holen
710            jsr $aefd       ;auf komma testen
720            jsr gtsnr       ;position innerhalb matrix
730            jsr $aefd       ;auf komma testen
740            jsr $b79e       ;matrixcode
750            txa
760            ldy zw          ;position innerhalb matrix
770            sta ($14),y     ;code in zeichensatz ablegen
780            rts             ;fertig
```

Damit Sie sich nicht mit irgendwelchen Speicheradressen herumschlagen müssen, können Sie bei allen Routinen die Nummer des gewünschten Zeichens angeben. Diese Nummer können Sie aus der Tabelle weiter oben entnehmen. Um beispielsweise die Zeichenmatrix des großen "A", das die Nummer 1 hat, auszulesen, gehen Sie wie folgt vor:

```
100 for mp=1 to 8
110 sys 51736,1,mp:mt$=mt$+chr$(peek(251))
120 next mp
```

Die Variable MT$ enthält anschließend die Zeichenmatrix, die Sie dann unmittelbar mit dem Zeichen-Editor darstellen und gegebenenfalls verändern können. Die folgenden Routinen manipulieren die Zeichenmatrix direkt im Zeichensatz:

```
790 ;
792 ;
794 ;*********************************************************
796 ;*                                                       *
798 ;* chmove - matrix eines zeichens verschieben            *
800 ;*                                                       *
802 ;* aufruf: sys 51801,z1,z2                               *
804 ;*      z1: zeichencode (quelle)                         *
806 ;*      z2: zeichencode (ziel)                           *
```

```
808 ;*                                                          *
810 ;************************************************************
815 ;
820 chmove      jsr chtest      ;zeichensatz testen
830             jsr $aefd       ;auf komma testen
840             jsr gtchcd2     ;zeichencodes holen
850             lda zp3         ;zeiger umspeichern
860             ldy zp4
870             sta $22
880             sty $23
890 chmv1       jsr ramkon      ;auf ram umschalten
900             ldy #0          ;daten im speicher ablegen
910 chmv2       lda ($22),y
920             sta ($14),y
930             iny
940             cpy #8
950             bne chmv2
960             jmp orgkon      ;auf rom zurueckschalten
970 ;
972 ;
974 ;************************************************************
976 ;*                                                          *
978 ;* chchange - zwei zeichenmatrizen vertauschen              *
980 ;*                                                          *
982 ;* aufruf: sys 51835,z1,z2                                  *
984 ;*  z1,z2: codes (0-511) der zu vertauschenden              *
986 ;*         zeichenmatrizen                                  *
988 ;*                                                          *
990 ;************************************************************
995 ;
1000 chchange jsr chgm       ;zum gemeinsamen teil
1010          ldy #0         ;datenblocks austauschen
1020 chh1     lda (zp3),y
1030          tax
1040          lda ($14),y
1050          sta (zp3),y
1060          txa
1070          sta ($14),y
1080          iny
1090          cpy #8
1100          bne chh1
1110          jmp orgkon     ;auf rom zurueckschalten
1120 ;
1150 ;*** gemeinsamer teil von chchange u. a. ***
1160 chgm     jsr chtest     ;zeichentest testen
1170          jsr $aefd      ;auf komma testen
1180          jsr gtchcd2    ;zeichencodes holen
1190          jmp ramkon     ;auf ram umschalten
1200 ;
1202 ;
1204 ;************************************************************
1206 ;*                                                          *
1208 ;* chinvert - zeichenmatrix invertieren                     *
```

```
1210 ;*                                                    *
1212 ;* aufruf: sys 51870,zn                                *
1214 ;*       zn: zeichencode (0-511)                       *
1216 ;*                                                    *
1218 ;******************************************************
1220 ;
1230 chinvert jsr chtest      ;zeichensatz testen
1240          jsr $aefd       ;auf komma testen
1250          jsr gtchrcd     ;zeichencode holen
1260          jsr ramkon      ;auf ram umschalten
1270          ldy #0          ;datenblock invertieren
1280 chi1     lda ($14),y
1290          eor #$ff
1300          sta ($14),y
1310          iny
1320          cpy #8
1330          bne chi1
1340          jmp orgkon      ;auf rom zurueckschalten
1350 ;
1352 ;
1354 ;******************************************************
1356 ;*                                                    *
1358 ;* chand - zwei zeichenmatrizen und-verknuepfen        *
1360 ;*                                                    *
1362 ;* aufruf: sys 51898,z1,z2                             *
1364 ;*  z1,z2: codes (0-511) der zu verknuepfenden         *
1366 ;*         zeichenmatrizen                             *
1368 ;*                                                    *
1370 ;******************************************************
1375 ;
1380 chand    jsr chgm        ;zum gemeinsamen teil
1390          ldy #0
1400 cha1     lda (zp3),y
1410          and ($14),y
1420          sta (zp3),y
1430          iny
1440          cpy #8
1450          bne cha1
1460          jmp orgkon      ;auf rom zurueckschalten
1470 ;
1472 ;
1474 ;******************************************************
1476 ;*                                                    *
1478 ;* chor - zwei zeichenmatrizen oder-verknuepfen        *
1480 ;*                                                    *
1482 ;* aufruf: sys 51917,z1,z2                             *
1484 ;*  z1,z2: codes (0-511) der zu verknuepfenden         *
1486 ;*         zeichenmatrizen                             *
1488 ;*                                                    *
1490 ;******************************************************
1495 ;
1500 chor     jsr chgm        ;zum gemeinsamen teil
1510          ldy #0
```

```
1520 cho1     lda (zp3),y
1530          ora ($14),y
1540          sta (zp3),y
1550          iny
1560          cpy #8
1570          bne cho1
1580          jmp orgkon    ;auf rom zurueckschalten
1590 ;
1592 ;
1594 ;*******************************************************
1596 ;*                                                     *
1598 ;* cheor - zwei zeichenmatrizen exclusiv-oder-verk.    *
1600 ;*                                                     *
1602 ;* aufruf: sys 51936,z1,z2                             *
1604 ;*  z1,z2: codes (0-511) der zu verknuepfenden         *
1606 ;*         zeichenmatrizen                             *
1608 ;*                                                     *
1610 ;*******************************************************
1615 ;
1620 cheor    jsr chgm      ;zum gemeinsamen teil
1630          ldy #0
1640 che1     lda (zp3),y
1650          eor ($14),y
1660          sta (zp3),y
1670          iny
1680          cpy #8
1690          bne che1
1700          jmp orgkon    ;auf rom zurueckschalten
```

Haben Sie alle Änderungen durchgeführt, müssen Sie dem Rechner noch mitteilen, daß er den neuen Zeichensatz benutzen soll. Dazu dient die Routine CHSWITCH:

```
1710 ;
1712 ;
1714 ;*******************************************************
1716 ;*                                                     *
1718 ;* chswitch - auf neuen zeichensatz umschalten         *
1720 ;*                                                     *
1722 ;* aufruf: sys 51955,nr                                *
1724 ;*     nr: nummer des zeichensatzes                    *
1726 ;*         (0-6; siehe charset)                        *
1728 ;*                                                     *
1730 ;*******************************************************
1735 ;
1740 chswitch jsr $aefd     ;auf komma testen
1750          jsr gtchsnr   ;zeichensatznummer nach x
1760          lda 53272     ;1. register
1770          and #240
1780          ora chrtab3,x ;entspr. bits setzen
1790          sta 53272
```

```
1800          lda 56576     ;2. register
1810          and #252
1820          ora chrtab2,x ;entspr. bits setzen
1830          sta 56576
1840          rts           ;fertig
1850 ;
1852 ;
1854 ;*********************************************************
1856 ;*                                                       *
1858 ;* choldcon - auf originalzeichensatz zuruecksch.        *
1860 ;*                                                       *
1862 ;* aufruf: sys 51984                                     *
1864 ;*                                                       *
1866 ;*********************************************************
1870 ;
1880 choldcon lda #21       ;1. register
1890          sta 53272
1900          lda 56576     ;2. register
1910          and #252
1920          ora #3
1930          sta 56576
1940          lda #4        ;3. register
1950          sta 648
1960          rts           ;fertig
```

Um wieder auf den "Originalzeichensatz" zurückzuschalten, benutzen Sie die Routine CHOLDCON. Assembler-Programmierer vergessen bitte nicht, noch die gemeinsamen Unterprogramme abzutippen. Die BASIC-Programmierer finden im Anschluß wieder einen BASIC-Lader zu allen Routinen:

```
1970 ;
1980 ;
1990 ;
2000 ;
2010 ;
2015 ;*********************************************************
2020 ;*                                                       *
2025 ;*   gemeinsame unterprogramme                           *
2030 ;*                                                       *
2035 ;*********************************************************
2040 ;
2060 ;*** zeichensatznummer einlesen ***
2070 gtchsnr  jsr $b79e     ;nummer einlesen
2080          cpx #7        ;>6?
2090          bcc gtc1      ;nein, ok
2100 gtcf     jmp $b248     ;ja, fehler
2110 gtc1     rts           ;fertig
2120 ;
2140 ;*** zeichencode einlesen und adresse berechnen ***
2150 gtchrcd  jsr $ad8a     ;nummer einlesen
```

```
2160          jsr $b7f7      ;und nach integer wandeln
2170          cmp #2         ;high-byte>1?
2180          bcs gtcf       ;ja, fehler
2190 gtchrcd2 ldx #3         ;zeichencode*8
2200 gtcd1    asl $14
2210          rol $15
2220          dex
2230          bne gtcd1
2240          clc            ;absolute adresse berechnen
2250          lda chadrl     ;basisadresse
2260          adc $14
2270          sta $14
2280          lda chadrh
2290          adc $15
2300          sta $15
2310          rts            ;fertig
2320 ;
2340 ;*** zwei zeichencodes bearbeiten ***
2350 gtchcd2  jsr gtchrcd    ;1. zeichencode holen
2360          lda $14        ;und umspeichern
2370          ldy $15
2380          sta zp3
2390          sty zp4
2400          jsr $aefd      ;auf komma testen
2410          jmp gtchrcd    ;2. zeichencode holen
2420 ;
2440 ;*** aktuellen zeichensatz testen ***
2450 chtest   lda chaklfl    ;flag laden
2460          bne chtt1
2470          jmp $b248      ;originalzeichensatz, fehler
2480 chtt1    rts            ;fertig
2490 ;
2510 ;*** matrixposition einlesen ***
2520 gtsnr    jsr $b79e      ;position
2530          txa            ;=0?
2540          bne gts1       ;nein, ok
2550 gts2     jmp $b248      ;ja, fehler
2560 gts1     cpx #9         ;>8?
2570          bcs gts2       ;ja, fehler
2580          dex
2590          stx zw         ;zwischenspeichern
2600          rts            ;fertig
2610 ;
2620 ;*** auf ram umschalten ***
2630 ramkon   sei            ;interrupt sperren
2640          lda #53
2650          sta $01
2660          rts            ;fertig
2670 ;
2690 ;*** auf rom umschalten ***
2700 orgkon   lda #55
2710          sta $01
2720          cli            ;interrupt freigeben
```

```
2730          rts              ;fertig
2740 ;
2750 ;
2760 ;
2770 ;*** zeichensatz-tabellen ******************************
2790 chrtab1  .wo 53248,0,4096,8192,12288,57344,61440
2810 chrtab2  .by 3,3,3,3,3,0,0   ;register 56576
2820 chrtab3  .by 4,0,4,8,12,8,12 ;register 53272
2830 ;
2840 ;
2850 ;
2860 ;*** datenspeicher ************************************
2870 chaklfl  .by 0        ;arbeitszeichensatz
2880 chadrl   .by 0        ;zeichensatzbasisadresse (low)
2890 chadrh   .by 0        ;(high)
```

Der BASIC-Lader:

```
100 rem *****************************************************
102 rem *                                                   *
104 rem * programm: zeichensatz                             *
106 rem * programmlaenge: 426 bytes                         *
108 rem * routinen zur zeichensatzprogrammierung            *
110 rem *                                                   *
112 rem *****************************************************
114 rem *                                                   *
116 rem * charset - aktuellen zeichensatz festlegen         *
118 rem *                                                   *
120 rem * aufruf: sys 51711,nr                              *
122 rem *     nr: nummer des zeichensatzes                  *
124 rem *         0: 53248-57343 (original!)                *
126 rem *         1:      0- 4095                           *
128 rem *         2   4096- 8191                            *
130 rem *         3   8192-12287                            *
132 rem *         4  12288-16383                            *
134 rem *         5  57344-61439                            *
136 rem *         6  61440-65535                            *
138 rem *                                                   *
140 rem *****************************************************
142 rem *                                                   *
144 rem * chread - matrix-byte eines zeichens auslesen      *
146 rem *                                                   *
148 rem * aufruf: sys 51736,zn,mp                           *
150 rem *     zn: zeichencode (0-511)                       *
152 rem *     mp: matrixposition (1-8)                      *
154 rem *                                                   *
156 rem *****************************************************
158 rem *                                                   *
160 rem * chwrite - matrix-byte eines zeichens schreiben    *
162 rem *                                                   *
164 rem * aufruf: sys 51774,zn,mp,wt                        *
166 rem *     zn: zeichencode (0-511)                       *
```

```
168 rem *     mp: matrixposition (1-8)                      *
170 rem *     wt: matrixcode (0-255)                        *
172 rem *                                                   *
174 rem *****************************************************
176 rem *                                                   *
178 rem * chmove - matrix eines zeichens verschieben        *
180 rem *                                                   *
182 rem * aufruf: sys 51801,z1,z2                           *
184 rem *     z1: zeichencode (quelle)                      *
186 rem *     z2: zeichencode (ziel)                        *
188 rem *                                                   *
190 rem *****************************************************
192 rem *                                                   *
194 rem * chchange - zwei zeichenmatrizen vertauschen       *
196 rem *                                                   *
198 rem * aufruf: sys 51835,z1,z2                           *
200 rem *  z1,z2: codes (0-511) der zu vertauschenden       *
202 rem *         zeichenmatrizen                           *
204 rem *                                                   *
206 rem *****************************************************
208 rem *                                                   *
210 rem * chinvert - zeichenmatrix invertieren              *
212 rem *                                                   *
214 rem * aufruf: sys 51870,zn                              *
216 rem *     zn: zeichencode (0-511)                       *
218 rem *                                                   *
220 rem *****************************************************
222 rem *                                                   *
224 rem * chand - zwei zeichenmatrizen und-verknuepfen      *
226 rem *                                                   *
228 rem * aufruf: sys 51898,z1,z2                           *
230 rem *  z1,z2: codes (0-511) der zu verknuepfenden       *
232 rem *         zeichenmatrizen                           *
234 rem *                                                   *
236 rem *****************************************************
238 rem *                                                   *
240 rem * chor - zwei zeichenmatrizen oder-verknuepfen      *
242 rem *                                                   *
244 rem * aufruf: sys 51917,z1,z2                           *
246 rem *  z1,z2: codes (0-511) der zu verknuepfenden       *
248 rem *         zeichenmatrizen                           *
250 rem *                                                   *
252 rem *****************************************************
254 rem *                                                   *
256 rem * cheor - zwei zeichenmatrizen exclusiv-oder-verk.  *
258 rem *                                                   *
260 rem * aufruf: sys 51936,z1,z2                           *
262 rem *  z1,z2: codes (0-511) der zu verknuepfenden       *
264 rem *         zeichenmatrizen                           *
266 rem *                                                   *
268 rem *****************************************************
270 rem *                                                   *
272 rem * chswitch - auf neuen zeichensatz umschalten       *
```

```
274 rem *                                                *
276 rem * aufruf: sys 51955,nr                           *
278 rem *      nr: nummer des zeichensatzes              *
280 rem *          (0-6; siehe charset)                  *
282 rem *                                                *
284 rem **************************************************
286 rem *                                                *
288 rem * choldcon - auf originalzeichensatz zuruecksch. *
290 rem *                                                *
292 rem * aufruf: sys 51984                              *
294 rem *                                                *
296 rem **************************************************
298 :
300 :
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=51711:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 32,253,174,32,37,203,142,166,203,138,1380
1010 data 10,168,185,138,203,141,167,203,185,139,1539
1020 data 203,141,168,203,96,32,253,174,32,48,1350
1030 data 203,32,253,174,32,109,203,120,169,53,1348
1040 data 174,166,203,208,2,169,51,133,1,164,1271
1050 data 2,177,20,133,2,32,132,203,164,2,867
1060 data 132,251,96,32,100,203,32,253,174,32,1305
1070 data 48,203,32,253,174,32,109,203,32,253,1339
1080 data 174,32,158,183,138,164,2,145,20,96,1112
1090 data 32,100,203,32,253,174,32,83,203,165,1277
1100 data 253,164,254,133,34,132,35,32,126,203,1366
1110 data 160,0,177,34,145,20,200,192,8,208,1144
1120 data 247,76,132,203,32,146,202,160,0,177,1375
1130 data 253,170,177,20,145,253,138,145,20,200,1521
1140 data 192,8,208,241,76,132,203,32,100,203,1395
1150 data 32,253,174,32,83,203,76,126,203,32,1214
1160 data 100,203,32,253,174,32,48,203,32,126,1203
1170 data 203,160,0,177,20,73,255,145,20,200,1253
1180 data 192,8,208,245,76,132,203,32,146,202,1444
1190 data 160,0,177,253,49,20,145,253,200,192,1449
1200 data 8,208,245,76,132,203,32,146,202,160,1412
1210 data 0,177,253,17,20,145,253,200,192,8,1265
1220 data 208,245,76,132,203,32,146,202,160,0,1404
1230 data 177,253,81,20,145,253,200,192,8,208,1537
1240 data 245,76,132,203,32,253,174,32,37,203,1387
```

```
1250 data 173,24,208,41,240,29,159,203,141,24,1242
1260 data 208,173,0,221,41,252,29,152,203,141,1420
1270 data 0,221,96,169,21,141,24,208,173,0,1053
1280 data 221,41,252,9,3,141,0,221,169,4,1061
1290 data 141,136,2,96,32,158,183,224,7,144,1123
1300 data 3,76,72,178,96,32,138,173,32,247,1047
1310 data 183,201,2,176,242,162,3,6,20,38,1033
1320 data 21,202,208,249,24,173,167,203,101,20,1368
1330 data 133,20,173,168,203,101,21,133,21,96,1069
1340 data 32,48,203,165,20,164,21,133,253,132,1171
1350 data 254,32,253,174,76,48,203,173,166,203,1582
1360 data 208,3,76,72,178,96,32,158,183,138,1144
1370 data 208,3,76,72,178,224,9,176,249,202,1397
1380 data 134,2,96,120,169,53,133,1,96,169,973
1390 data 55,133,1,88,96,0,208,0,0,0,581
1400 data 16,0,32,0,48,0,224,0,240,3,563
1410 data 3,3,3,3,0,0,4,0,4,8,28
1420 data 12,8,12,0,0,0,0,0,0,0,32
2000 data -1:rem endmarkierung
```

Nachdem wir die Maschinensprache-Routinen erarbeitet haben, können wir das Hauptprogramm des Zeichen-Editors ergänzen. Als erstes fügen Sie die Zeile 146 ein, die die Routinensammlung nachlädt:

```
146 if fl=1 then fl=2:load "zeichensatz",8,1
```

Außerdem benötigen wir ja die Routine TRANSFER:

```
147 if fl=2 then fl=3:load "transfer",8,1
```

Um den geänderten Zeichensatz dauerhaft zu sichern, muß er auf Diskette gespeichert werden. Dazu verwenden wir die Routine DSAVEM aus Kapitel 4:

```
148 if fl=3 then fl=4:load "dsavem",8,1
```

Das weitere Hauptprogramm sieht wie folgt aus:

```
210 rem zeichensatz in bereich 12288-16383 verlagern
215 sys 49152,53248,57343,12288,1:rem umspeichern
220 sys 51711,4:rem charset
225 sys 51955,4:rem chswitch
230 :
235 print "neues zeichen entwerfen : -1-"
240 print "altes zeichen aendern:    -2-"
242 print "programm beenden:         -3-"
```

```
245 input "ihre wahl (1,2,3): ";wl
250 if wl<1 and wl>3 then 250:rem falsche eingabe
255 :
257 if wl=3 then end
260 if wl=2 then 335
265 rem neues zeichen
270 gosub 1250:rem editor
280 print chr$(147)
290 input "nummer des zeichens: ";zn
300 rem zeichenmatrix in zeichensatz schreiben (chwrite)
305 for mp=1 to 8
310 sys 51774,zn,mp,asc(mid$(mt$,mp,1))
315 next mp
320 goto 235:rem hauptmenue
325 :
330 rem altes zeichen
335 print chr$(147)
340 input "nummer des zeichens: ";zn
350 rem zeichenmatrix aus zeichensatz holen (chread)
352 mt$=""
355 for mp=1 to 8
360 sys 51736,zn,mp:mt$=mt$+chr$(peek(251))
365 next mp
370 gosub 1100:rem editor
375 rem zeichenmatrix in zeichensatz schreiben (chwrite)
380 for mp=1 to 8
385 sys 51774,zn,mp,asc(mid$(mt$,mp,1))
390 next mp
395 goto 235:rem hauptmenue
```

Das Programm fragt Sie, ob Sie ein neues Zeichen entwerfen oder ein vorhandenes Zeichen darstellen und verändern wollen. Ein neues Zeichen wird anschließend an der erfragten Position (Nummer des Zeichens) in den Zeichensatz geschrieben.

Im zweiten Fall möchte das Programm die Nummer des Zeichens wissen, dessen Zeichenmatrix dargestellt werden soll. Nachdem Sie die Änderungen durchgeführt haben, wird die geänderte Zeichenmatrix in den Zeichensatz zurückgeschrieben. Um den geänderten Zeichensatz zum Schluß auf Diskette zu speichern, geben Sie einfach ein:

```
SYS 49489,"ZEICHENSATZ",12288,16383:REM DSAVEM
```

Wie eingangs versprochen, möchte ich Ihnen zum Schluß ein Programm vorstellen, das einen "deutschen Zeichensatz" erzeugt: Die Buchstaben Y und Z werden vertauscht. Anstelle des

"Pfund"-Zeichens tritt ein Doppel-S (ß). Die deutschen Umlaute ä, ö und ü sind über die Tasten für den Doppelpunkt, das Semikolon sowie den "Klammeraffen" erreichbar.

```
100 rem deutscher zeichensatz auf dem 64'er
110 :
120 rem maschinenprogramme nachladen
130 if fl=0 then fl=1:load "transfer",8,1
140 if fl=1 then fl=2:load "zeichensatz",8,1
150 :
160 rem zeichensatz in bereich 12288-16383 verlagern
170 sys 49152,53248,57343,12288,1:rem umspeichern
180 sys 51711,4:rem charset
190 sys 51955,4:rem chswitch
200 :
210 rem y und z vertauschen (mit chchange)
220 sys 51835,25,26:rem teilzeichensatz 1, grossbuchstaben
230 sys 51835,153,154:rem teilzeichensatz 1, grossb., revers
240 sys 51835,281,282:rem teilzeichensatz 2, kleinbuchstaben
250 sys 51835,409,410:rem teilzeichensatz 2, kleinb., revers
260 sys 51835,345,346:rem teilzeichensatz 2, grossbuchstaben
270 sys 51835,473,474:rem teilzeichensatz 2, grossb., revers
280 :
290 rem deutscher umlaut 'ä'
300 sys 51801,1,59:rem 'a' zum ';' (chmove)
310 sys 51774,59,1,219:rem punkte einfuegen (chwrite)
320 sys 51801,59,187:rem (chmove) inverses 'ä'
330 sys 51870,187:rem (chinvert)  erzeugen
340 sys 51801,257,315:rem (chmove) teilzeichensatz 2
350 sys 51774,315,1,102:rem (chwrite)
360 sys 51801,315,443:rem (chmove)
370 sys 51870,443:rem (chinvert)
380 sys 51801,59,285:rem (chmove)
390 sys 51801,187,413:rem (chmove)
400 :
410 rem deutscher umlaut 'ö'
420 sys 51801,15,58:rem 'o' zum ':' (chmove)
430 sys 51774,58,1,195:rem punkte    (chwrite)
440 sys 51774,58,2,60:rem  einfuegen (chwrite)
450 sys 51801,58,186:rem (chmove) inverses 'ö'
460 sys 51870,186:rem (chinvert)  erzeugen
470 sys 51801,271,314:rem (chmove) teilzeichensatz 2
480 sys 51774,314,1,102:rem (chwrite)
490 sys 51801,314,442:rem (chmove)
500 sys 51870,442:rem (chinvert)
510 sys 51801,58,284:rem (chmove)
520 sys 51801,186,412:rem (chmove)
530 :
540 rem deutscher umlaut 'ü'
550 sys 51801,21,0:rem 'u' zum '@' (chmove)
560 sys 51774,0,2,0:rem punkte einfuegen (chwrite)
570 sys 51801,0,128:rem (chmove) inverses 'ä'
```

```
580 sys 51870,128:rem (chinvert)  erzeugen
590 sys 51801,277,256:rem (chmove) teilzeichensatz 2
600 sys 51774,256,3,0:rem (chwrite)
610 sys 51774,256,1,102:rem (chwrite)
620 sys 51801,256,384:rem (chmove)
630 sys 51870,384:rem (chinvert)
640 sys 51801,0,378:rem (chmove)
650 sys 51801,128,506:rem (chmove)
660 :
670 rem 'doppel-s'
680 for mp=1 to 8
690 read mc:sys 51774,28,mp,mc:rem (chwrite)
700 next mp
710 sys 51801,28,156:rem (chmove)
720 sys 51870,156:rem (chinvert)
730 sys 51801,28,284:rem (chmove)
740 sys 51801,156,412:rem (chmove)
750 data 60,102,102,124,102,102,124,96
```

6.5 Einführung in die GEOS-Programmierung

Ein weiteres Betätigungsfeld für grafikbegeisterte Programmierer bietet die Benutzeroberfläche GEOS. Leider ist GEOS aber nicht für eine Zusammenarbeit mit dem BASIC 2.0 ausgelegt.

Möchte man unter GEOS programmieren, kommt man daher um Assembler nicht herum. Um eigene Programme unter GEOS schreiben zu können, müssen wir uns zunächst einmal mit dem grundsätzlichen Aufbau von GEOS befassen. GEOS besteht an sich aus vier "Teilen":

- dem GEOS-Kernal
- dem Desktop
- den Applikationen
- den Accessoires

Das Desktop, der "Schreibtisch" von GEOS, dürfte Ihnen zur Genüge vertraut sein. Bei den Applikationen handelt es sich um Anwendungsprogramme, wie etwa GEOWRITE oder GEO-PAINT, die vom Desktop aus gestartet werden können.

Das besondere bei den Accessoires ist die Möglichkeit, sie aus einer Applikation heraus aufrufen zu können. Bekannte Beispiele

für Accessoires sind das Notepad, die Alarm-Clock und der Calculator. Sowohl das Desktop als auch sämtliche Applikationen und Accessoires arbeiten mit denselben gestalterischen Elementen. Dazu zählen insbesondere die sogenannten Pulldown-Menüs und die Dialogboxen.

Wie Sie sich leicht vorstellen können, erfordert es einiges an Programmieraufwand, um etwa eine Dialogbox oder ein Pulldown-Menü zu erzeugen. Daher wäre es natürlich reine Speicherplatzverschwendung, wenn jede Applikation und jedes Accessory die Programmroutinen zur Erzeugung dieser Elemente enthalten würde.

Aus diesem Grund sind alle Routinen, die von allgemeiner Bedeutung sind, in einem eigenen Programm zusammengefaßt, dem GEOS-Kernal. Das GEOS-Kernal stellt sozusagen das "Betriebssystem" von GEOS dar. Es ist im Grunde genommen nichts anderes als eine Sammlung von Unterprogrammen, die von den anderen GEOS-Teilen genutzt werden. Wenn eine Applikation beispielsweise einen Text ausgeben möchte, ruft sie einfach mit

```
JSR $.....
```

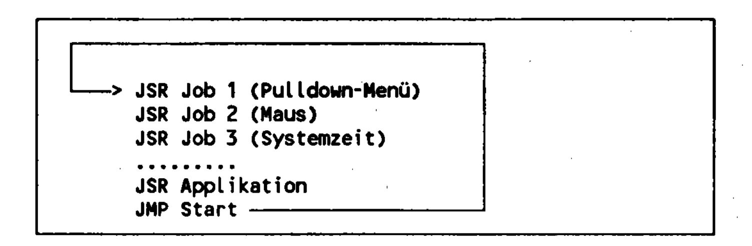
das entsprechende Unterprogramm im GEOS-Kernal auf. Diese Vorgehensweise kennen Sie ja auch schon vom "normalen" Betriebssystem des Commodore 64, das ja ebenfalls verschiedenste Routinen, etwa zur Ein- und Ausgabe von Daten, bereitstellt, mit denen man sich das Programmiererleben leichter machen kann.

Das GEOS-Kernal kann aber noch mehr. Wenn ein Programm ein Pulldown-Menü anbietet, muß dieses ständig "überwacht" werden, da ja der Anwender des Programms die Möglichkeit hat, einen bestimmten Menüpunkt zu (fast) jedem beliebigen Zeitpunkt anzuklicken.

Zu diesem Zweck verfügt das GEOS-Kernal über eine zentrale "Job-Schleife", die regelmäßig durchlaufen wird. Innerhalb dieser Job-Schleife werden verschiedene Unterprogramme aufgerufen, die beispielsweise kontrollieren, ob ein Menüpunkt eines

Pulldown-Menüs angeklickt wurde, und unter anderem dafür sorgen, daß die interne Uhr von GEOS weitergestellt wird.

Die jeweils gerade ablaufende Applikation wird ebenfalls als ein Unterprogramm in die Job-Schleife "eingehängt" und von dieser immer wieder aufgerufen. Grafisch veranschaulicht sieht das so aus:



Tatsächlich ist der Aufbau der Job-Schleife etwas komplizierter. Hier soll es ja aber nur um das Prinzip gehen. Wesentlich ist, daß die Kontrolle der Systemaktivitäten unter GEOS nicht wie sonst beim Anwendungsprogramm liegt, sondern beim GEOS-Kernal, das je nach aufgetretenem Ereignis (z.B. Mausklick) in entsprechende Unterprogramme verzweigt.

Es würde an dieser Stelle zu weit führen, auf weitere Details, etwa, wie eine selbstgeschriebene Applikation in die Job-Schleife eingebunden werden muß, einzugehen. Ich möchte Ihnen hier aber trotzdem einmal einen kurzen praktischen Einblick in die Programmierung unter GEOS geben. Alle Routinen des GEOS-Kernal sind - wie die des 64'er-Betriebssystems - über eine sogenannte "Sprungtabelle" erreichbar.

Diese Sprungtabelle liegt im Rechnerspeicher von $C100 bis $C2C7. Die Routine zur Erzeugung von Dialogboxen jeder Art beispielsweise hat die Adresse $C256, kann also mit

```
JSR $C256
```

aufgerufen werden. Nehmen wir einmal an, Sie möchten in einem Programm eine Dialogbox erzeugen. Wie sieht das nun kon-

kret aus? Die Dialogbox-Routine benötigt natürlich Informationen darüber, wie die Dialogbox aussehen soll. Diese werden ihr in einer Datentabelle übergeben. Wo die Tabelle im Rechnerspeicher liegt, teilen Sie der Routine über einen sogenannten "Zeiger" mit, der in den Speicherzellen $02/$03 übergeben wird:

```
100           lda #<(tabelle) ;zeiger
110           ldy #>(tabelle) ;auf tabelle
120           sta $02         ;zeiger
130           sty $03         ;setzen
140           jsr $c256       ;routine aufrufen
.........
500 tabelle   .by .....
```

Durch gezielte Änderungen in der Datentabelle lassen sich so völlig verschiedenartige Dialogboxen erzeugen, ohne am Programm selbst etwas ändern zu müssen.

Die Tabelle nun muß wie folgt aufgebaut sein: Das erste Byte der Tabelle bestimmt die Größe der Dialogbox. Ist das 7. Bit des Bytes gleich 1, wird ein Standardfenster, wie es auch im Desktop verwendet wird, erzeugt. Ist das Bit gleich Null, müssen in den folgenden sechs Bytes die genauen Koordinaten der Dialogbox folgen:

- oberer Rand
- unterer Rand
- linker Rand (low)
- linker Rand (high)
- rechter Rand (low)
- rechter Rand (high)

Zur Textaus- und -eingabe und für die sogenannten Klickfelder stehen spezielle Kommandocodes zur Verfügung. Der Code $0B beispielsweise gibt einen Text in die Dialogbox aus. Dazu werden hinter dem Code folgende Daten abgelegt:

- Abstand des Textes vom linken Rand des Fensters
- Abstand vom oberen Rand
- Zeiger auf den auszugebenden Text (low)
- Zeiger auf den auszugebenden Text (high)

Der auszugebende Text (und auch die Datentabelle) müssen mit einer Null abgeschlossen sein. Ein Beispiel: Sie möchten einen Text in ein Fenster im oberen Bereich des Bildschirms ausgeben lassen. Die Tabelle muß dann so aufgebaut sein:

```
500 tabelle   .by $01         ;bit7=0 ->koordinaten folgen
510           .by $0a         ;oberer Rand (y=10)
520           .by $32         ;unterer Rand (y=50)
530           .by $0a         ;linker Rand (x=10)
540           .by $00         ;high-byte
550           .by $2c         ;rechter Rand (x=300)
560           .by $01         ;high-byte
570           .by $0b         ;textausgabe
580           .by $10         ;abstand vom linken rand
590           .by $10         ;abstand vom oberen rand
600           .by <(text)     ;zeiger
610           .by >(text)     ;auf text
620           .by $00         ;tabellenende
630 ;
640 text      .tx "das ist ein beispieltext."
650           .by 0   ;textende
```

Für die sechs Klickfelder sind die folgenden Kommandocodes zuständig:

```
$01: OK
$02: CANCEL
$03: YES
$04: NO
$05: OPEN
$06: DISK
```

Hinter dem Kommandocode folgt in zwei Bytes der Abstand des Feldes vom linken und oberen Rand. Um beispielsweise eine Ja/ Nein-Abfrage zu realisieren, geht man so vor:

```
100           lda #<(tabelle) ;zeiger
110           ldy #>(tabelle) ;auf tabelle
120           sta $02         ;zeiger
130           sty $03         ;setzen
140           jsr $c256       ;routine aufrufen
150           lda $02         ;$02 enthaelt kommandocode
160                           ;des geklickten feldes
170           cmp #$03        ;'yes'-feld geklickt?
180           beq yes         ;ja
190           cmp #$04        ;'no'-feld geklickt?
200           beq no          ;ja
......
```

```
300 yes         .....          ;'yes'-feld
......
400 no          .....          ;'no'-feld
.......
500 tabelle     .by $81        ;bit7=1 ->standardfenster
510             .by $0b        ;textausgabe
520             .by $05        ;abstand vom linken rand
530             .by $0a        ;abstand vom oberen rand
540             .by <(text)    ;zeiger
550             .by >(text)    ;auf text
560             .by $03        ;'yes'-feld
570             .by $05        ;abstand vom linken rand
580             .by $40        ;abstand vom oberen rand
590             .by $04        ;'no'-feld
600             .by $15        ;abstand vom linken rand
610             .by $40        ;abstand vom oberen rand
620             .by $00        ;tabellenende
630 ;
640 text        .tx "wollen sie weitermachen?"
650             .by 0   ;textende
```

Wie Sie sehen, läßt sich unter GEOS sehr komfortabel und flexibel programmieren. Aber auch ohne GEOS sind die Möglichkeiten der Grafikprogrammierung auf dem Commodore 64 derart vielfältig, daß ich Ihnen im Rahmen dieses Kapitels unmöglich alles vorstellen und zeigen konnte.

7. Sound und Musik

In diesem Kapitel geht es um die Programmierung von Sound und Musik auf dem Commodore 64. Auch dabei steht die Praxis wieder im Mittelpunkt. Die Theorie habe ich auf das wirklich notwendige beschränkt. Leider läßt einen das BASIC 2.0 auch bei der Sound-Programmierung fast vollkommen im Stich. Der einzige Weg führt über unhandliche POKE- und PEEK-Wüsten. Da man damit natürlich nicht vernünftig arbeiten kann, habe ich für Sie wieder eine umfangreiche Sammlung von Mashinensprache-Routinen zusammengestellt, die die Programmierung wesentlich vereinfachen.

Zunächst sollten Sie aber einmal die wichtigsten Grundlagen zur Sound-Programmierung erfahren.

7.1 Grundlagen

Verantwortlich für die Tonerzeugung ist ein spezieller Baustein im Commodore 64, der sogenannte "Sound Interface Device", kurz SID. Möchte man dem Commodore 64 einen bestimmten Ton entlocken, muß man den SID entsprechend programmieren.

Dazu verfügt der SID über insgesamt 24 Register, die allerdings nur beschrieben und nicht gelesen werden können. In den einzelnen Registern hat zum Teil jedes Bit eine verschiedene Funktion. Der SID verfügt über drei getrennte Tongeneratoren, die in einem Frequenzbereich von null bis etwa 4.000 Hertz arbeiten können. Jeder der Tongeneratoren kann unabhängig von den anderen über jeweils sieben Register gesteuert werden. Der Commodore 64 ist also in der Lage "dreistimmig" zu spielen.

Über die Register lassen sich die gewünschte Frequenz eines Tones, seine Wellenform sowie die sogenannte Hüllkurve, die den Lautstärkeverlauf beeinflußt, einstellen. Zusätzlich verfügt der SID über einen sogenannten Filter mit dem sich die Ausgaben der drei Tongeneratoren auf vielfältige Art und Weise verändern

lassen. Bei jedem der Tongeneratoren kann der Filter wahlweise zugeschaltet werden. Damit wäre der Grobaufbau des SID auch schon geklärt. Es gibt also drei voneinander unabhängige Tongeneratoren, oft auch "Stimmen" genannt, und einen wahlweise zuschaltbaren Filter. Schön und gut, doch bevor wir uns nun konkret ans Programmieren machen, sollten wir zunächst kurz die Frage klären, was überhaupt ein Ton ist.

7.2 Etwas Tontheorie

Wie Sie sicher wissen, sind Töne, Musik und Geräusche nichts anderes als mehr oder minder komplexe Luftschwingungen, die im menschlichen Ohr vom Trommelfell wahrgenommen werden. Jede Luftschwingung läßt sich physikalisch als Welle auffassen. Die beiden wesentlichen Kriterien zum Beschreiben einer Welle (und damit eines Tones) sind ihre Frequenz und ihre Amplitude.

Die Frequenz bezeichnet die Anzahl der Schwingungen je Sekunde. Ihre Einheit ist "Hertz". Bei einem Ton mit der Frequenz 1.000 Hertz finden also tausend Schwingungen je Sekunde statt. Je höher die Frequenz ist, desto höher empfinden wir den betreffenden Ton. Die Tongeneratoren des SID sind in der Lage, Töne mit einer Frequenz zwischen 0 und etwa 4.000 Hertz zu erzeugen.

Die Amplitude bezeichnet die Höhe des "Ausschlags" einer Schwingung. Je größer die Amplitude ist, desto "lauter" wird der betreffende Ton empfunden. Beim SID läßt sich die Lautstärke nur global zwischen Werten von 0 bis 15 einstellen, wobei 15 für die Maximal-Lautstärke steht.

Ein weiterer wichtiger Punkt ist die "Form" der Welle. Der SID kann vier grundlegende Wellenformen erzeugen: Dreieck, Sägezahn, Rechteck und Rauschen. Töne in der Natur setzen sich meistens aus Schwingungen der verschiedensten Formen zusammen. Der SID ist nur in der Lage jeweils eine bestimmte Wellenform zu erzeugen. Nicht zuletzt deshalb klingen die Töne aus

dem Computer irgendwie "unnatürlich". Zwar kann man auch mehrere Wellenformen überlagern, das bringt aber in der Regel keine große Klangverbesserung.

Mit den drei Komponenten Lautstärke, Frequenz und Wellenform könnte man nun schon einfache Dauertöne erzeugen. Mehr aber auch nicht.

Was jetzt noch fehlt, ist die Möglichkeit, den Lautstärkeverlauf eines Tones festzulegen. Dazu dient - nicht nur auf dem Commodore 64, sondern ganz allgemein in der Tontheorie - die sogenannte ADSR-Hüllkurve. Durch diese Hüllkurve wird festgelegt, zu welchem Zeitpunkt der betreffende Ton welche Lautstärke haben soll. Der gesamte Lautstärkeverlauf wird dabei in vier Phasen unterteilt: Attack, Decay, Sustain und Release. Daher auch die Bezeichnung "ADSR".

Attack

In der Attack-Phase steigt die Lautstärke des Tones von null auf die Gesamtlautstärke an. Die Zeit dafür kann beim SID zwischen 0,002 und 8 Sekunden liegen.

Decay

Nachdem die Lautstärke ihren Maximalwert erreicht hat, beginnt die Decay-Phase. In dieser Phase fällt die Lautstärke wieder ab, allerdings nur auf den durch den Sustain-Parameter bestimmten Wert. Die Zeitspanne für die Decay-Phase darf zwischen 0,008 und 24 Sekunden liegen.

Sustain

In der Sustain-Phase bleibt die Lautstärke konstant auf dem festgelegten Wert. Die Zeitdauer der Sustain-Phase läßt sich beim SID nicht einstellen. Man muß sie "extern" durch eine Programmschleife festlegen.

Release

In der Release-Phase fällt die Lautstärke wieder bis auf Null ab. Die Zeitspanne für die Release-Phase darf zwischen 0,008 und 24 Sekunden liegen. Durch geschickte Wahl der Hüllkurve lassen sich die interessantesten Effekte erzielen. Schon kleinste Änderungen an den einzelnen Parametern bewirken völlig andere Klänge.

Mit diesem kleinen Exkurs haben wir alles an Theorie beieinander, was wir für die SID-Programmierung benötigen. Wenn Ihnen jetzt das eine oder andere noch nicht ganz klar ist, macht das natürlich überhaupt nichts. Bekanntlich lernt man ja durch die Praxis am besten, fangen wir also an!

7.3 Den Sound-Chip programmieren

Den Mittelpunkt dieses Abschnitts bildet eine Sammlung von 16 Mashinensprache-Routinen, mit denen sich alle Sound-Effekte sehr komfortabel programmieren lassen. Die einzelnen Routinen werde ich Ihnen im Laufe dieses Abschnitts nach und nach vorstellen. Wenn Sie die Routinen von Assembler aus nutzen wollen, beachten Sie bitte, daß die einzelnen Routinen nur als Gesamtprogramm lauffähig sind. Für BASIC-Programmierer befindet sich am Ende des Abschnitts wieder ein BASIC-Lader für das gesamte Programmpaket. Dort sind noch einmal alle Routinen mit ihren Aufrufparametern in REM-Zeilen zusammengefaßt.

Zunächst folgt der Programmkopf. Als Startadresse habe ich 40.000 gewählt. Leider haben die Routinen nicht mehr - wie die anderen - in den Speicherbereich hinter dem BASIC-Interpreter gepaßt. Deshalb habe ich sie notgedrungen am Ende des BASIC-Speichers untergebracht.

Bitte beachten Sie, daß das Ende des BASIC-Speichers aus diesem Grund auf die Adresse 39.999 herabgesetzt werden muß! Der BASIC-Lader am Ende des Abschnitts erledigt das automatisch. Wenn Sie das Programmpaket später von Diskette oder

Kassette nachladen lassen, sollte die erste Zeile in Ihrem Programm wie folgt aussehen:

```
10 poke 55,63:poke 56,156:clr
```

Erst im Anschluß daran kommt die Ladeanweisung. Wenn Sie diese Anweisungen vergessen, laufen Sie Gefahr, daß die Routinen durch String-Variablen überschrieben werden. Doch nun zum Programmcode:

```
100 ;*********************************************************
105 ;*                                                       *
110 ;* programm: sound                                       *
120 ;* routinen zur soundprogrammierung                      *
140 ;*                                                       *
150 ;*********************************************************
200 ;
205 ;
210 .ba 40000    ;*** startadresse ***
220 ;
230 ;
240 ;*** labels ********************************************
245 .gl zw  =  2   ;zwischenspeicher
250 .gl zp1 = 251
255 .gl zp2 = 252
260 .gl zp3 = 253
265 .gl zp4 = 254
270 ;
280 ;
```

Die erste Routine SDCLEAR dient dazu, sämtliche Register des SID zu löschen. Das ist besonders bei "mißglückten" Sound-Versuchen sehr nützlich. So müssen Sie beispielsweise nicht extra alle drei Tongeneratoren abschalten oder die Gesamtlautstärke zurückstellen. Aufgerufen wird SDCLEAR durch SYS 40000; Parameter sind nicht erforderlich.

Wie ich eingangs schon kurz erwähnt habe, können die SID-Register nur beschrieben werden. Aus diesem Grund werden die Register-Inhalte im hier vorgestellten Programmpaket zusätzlich in einem Zwischenspeicher abgelegt, von wo aus sie bei Bedarf bequem ausgelesen werden können.

Konkret sieht das so aus: Wenn ein Wert geändert werden soll, so geschieht das zunächst an der entsprechenden Stelle im Zwi-

schenspeicher. Anschließend wird immer der gesamte Zwischenspeicher in die SID-Register übertragen. Dieses Verfahren erweist sich am effizientesten, weil man nur eine Routine zum Übertragen der Inhalte benötigt.

Für alle, die es interessiert: Die Register des SID belegen im Speicher die Adressen 54.272 bis 54.296. Eine komplette Register-Tabelle finden Sie übrigens im Anhang.

Wenn Sie sich diese genauer anschauen, werden sie feststellen, daß der SID von Adresse 54.297 bis 54.300 noch über vier spezielle Lese-Register verfügt. Da sie aber für die Sound-Erzeugung nicht von Bedeutung sind, möchte ich in diesem Kapitel nicht auf sie eingehen. Der Programmcode von SDCLEAR:

```
200 ;*********************************************************
205 ;*                                                       *
210 ;* sdclear - sid-register loeschen                       *
220 ;*                                                       *
220 ;* aufruf: sys 40000                                     *
265 ;*                                                       *
270 ;*********************************************************
300 ;
340 sdclear   ldx #24        ;anzahl der register-1
350           lda #0         ;zwischenspeicher
360 sdcllp    sta sdzwsp,x   ;loeschen
370           dex
380           bpl sdcllp     ;alle register loeschen
390           jmp sdumsp     ;sid aktualisieren
400 ;
410 ;
```

Leider läßt der SID nur eine Gesamtlautstärke für alle drei Tongeneratoren zu. Eingestellt wird diese mit der Routine SDVOLUME. Es stehen insgesamt 16 mögliche Werte (0 bis 15) zur Verfügung. Der Wert Null bedeutet dabei, daß der Ton nicht hörbar ist, beim Wert 15 hat der Ton Maximal-Lautstärke.

Aufgerufen wird SDVOLUME mit SYS 40013,LS. LS steht für den gewünschten Lautstärkewert. SYS 40013,8 beispielsweise setzt die Lautstärke auf einen mittleren Wert. Der Programmcode von SDVOLUME:

```
412 ;**********************************************************
414 ;*                                                        *
416 ;* sdvolume - gesamtlautstaerke einstellen                *
418 ;*                                                        *
420 ;* aufruf: sys 40013,ls                                   *
422 ;*      ls: lautstaerke (0-15)                            *
424 ;*                                                        *
426 ;**********************************************************
428 ;
430 sdvolume   jsr $aefd      ;auf komma testen
440            jsr sdwtfq     ;wert holen und pruefen
450            lda sdzwsp+24  ;entspr. register holen
460            and #$f0
470            ora zw         ;lautstaerke setzen
480            sta sdzwsp+24
490            jmp sdumsp     ;sid aktualisieren
500 ;
510 ;
```

Die Frequenz läßt sich für die drei Tongeneratoren getrennt einstellen. Diese Aufgabe übernimmt die Routine SDFREQ. Die Frequenz darf sich im Bereich zwischen 0 und 3.848 Hertz bewegen. Je höher die Frequenz, desto höher der Ton.

Aufgerufen wird SDFREQ mit SYS 40032,TG,FR. TG steht für den gewünschten Tongenerator (1-3), FR für die Frequenz. Beispiel:

```
100 sys 40032,1,1000
110 sys 40032,2,2000
120 sys 40032,3,3000
```

Die Frequenz des Tongenerators 1 wird auf 1.000, die des Tongenerators 2 auf 2.000 und die des Tongenerators 3 auf 3.000 Hertz gesetzt. Die Frequenz kann übrigens jederzeit, während ein Ton gespielt wird, geändert werden. Dadurch lassen sich interessante Effekte erzielen. Der Programmcode von SDFREQ:

```
520 ;**********************************************************
522 ;*                                                        *
524 ;* sdfreq - frequenz einer stimme einstellen              *
526 ;*                                                        *
528 ;* aufruf: sys 40032,tg,fr                                *
530 ;*      tg: stimme (1-3)                                  *
532 ;*      fr: frequenz in Hertz (0-3848)                    *
534 ;*                                                        *
536 ;**********************************************************
```

```
538 ;
540 sdfreq     jsr $aefd        ;auf komma testen
545            jsr sdtgnr       ;stimmennr. holen
550            jsr $aefd        ;auf komma testen
560            jsr $ad8a        ;frequenz holen
570            lda #<(sdtab4)   ;zeiger auf
580            ldy #>(sdtab4)   ;konstante
590            jsr $ba28        ;mit frequenz multiplizieren
600            jsr $b7f7        ;nach 16-bit wandeln
610            ldy #0
620            lda $14          ;frequenz (low)
630            sta (zp1),y
640            iny
650            lda $15          ;(high)
660            sta (zp1),y
670            jmp sdumsp       ;sid aktualisieren
682 ;
684 ;
```

Wie Sie vielleicht wissen, entspricht jede Musiknote einer bestimmten Frequenz. Möchte man nun ein Musikstück spielen lassen, so müßte man die Noten extra in ihre zugehörigen Frequenzen umrechnen. Ein sehr mühseliges Unterfangen. Diese Aufgabe überläßt man besser einer entsprechenden Routine SD-NOTE. Bei SDNOTE genügt die Angabe der Note in der üblichen Notation. Der Aufruf erfolgt mit SYS 40.068,TG,NT$. TG gibt wieder die Nummer des gewünschten Tongenerators an. NT$ enthält die zu spielende Note in der Form "Note-Oktave". Mehr dazu im nächsten Abschnitt.

Ich wollte SDNOTE nur gleich an dieser Stelle ins Spiel bringen, da sie ja in unmittelbarem Zusammenhang mit der Frequenzangabe über SDFREQ steht. Hier der Programmcode:

```
686 ;*****************************************************
688 ;*                                                   *
690 ;* sdnote - musiknote eingeben                       *
692 ;*                                                   *
694 ;* aufruf: sys 40068,tg,nt$                          *
696 ;*      tg: stimme (1-3)                             *
698 ;*     nt$: musiknote ('c0'-'a#7')                   *
700 ;*                                                   *
702 ;*****************************************************
704 ;
710 sdnote     jsr $aefd        ;auf komma testen
720            jsr sdtgnr       ;stimmennr. holen
730            jsr $aefd        ;auf komma testen
740            jsr $ad9e        ;notenstring holen
```

```
750             jsr $b6a3
760 ;*** note holen ***
770             ldy #1
780             lda ($22),y
790             sta zw
800             dey
810             lda ($22),y
820             sta zp3
830             iny
840 ;*** note in tabelle suchen ***
850             ldx #0          ;zaehler
860 sdnt1       cmp sdtab10,x
870             beq sdnt2       ;gleich, dann weiter
880             inx
890             cpx #12         ;tabellenende?
900             bcc sdnt1       ;nein, weiter suchen
910 sdnterr     jmp $af08       ;ja, 'syntax error'
920 ;*** '#'-bearbeitung ***
930 sdnt2       lda zw          ;2. zeichen
940             cmp #35         ;'#'?
950             bne sdnt3       ;nein, weiter
960             inx
970             cmp sdtab10,x   ;'#' erlaubt?
980             bne sdnterr     ;nein, fehler
990             iny
1000            lda ($22),y     ;zeichen hinter '#'
1010            sta zw
1020 ;*** note in frequenz umrechnen ***
1030 sdnt3      txa
1040            asl
1050            tax
1060            lda sdtab11,x   ;basiswerte holen
1070            sta zp3
1080            inx
1090            lda sdtab11,x
1100            sta zp4
1110            lda zw          ;oktave
1120            sec
1130            sbc #$30
1140            cmp #8          ;>7?
1150            bcc sdnt4       ;nein, weiter
1160 sdnterr2   jmp $b248       ;ja, 'illegal quantity'
1170 sdnt4      eor #7          ;7. oktave?
1180            bne sdnt5       ;nein
1190            ldy #0          ;ja, note kontrollieren
1200            lda ($22),y
1210            cmp #$48        ;'h'?
1220            beq sdnterr2    ;ja, fehler
1230            jmp sdnt6       ;nein, weiter
1240 sdnt5      tax             ;zaehler
1250            ldy #0
1260            lda ($22),y     ;note
1270            cmp #$48        ;'h'?
```

```
1280         bne sdnt5a      ;nein
1290         sec             ;frequenzberechnung
1300         ror zp4
1310         bmi sdnt5b
1320 sdnt5a  lsr zp4
1330 sdnt5b  ror zp3
1340         dex
1350         bne sdnt5a
1360 sdnt6   ldy #0
1370         lda zp3
1380         sta (zp1),y
1390         iny
1400         lda zp4
1410         sta (zp1),y
1420         jmp sdumsp      ;sid aktualisieren
1430 ;
1432 ;
```

Für den Lautstärkeverlauf eines Tones ist, wie bereits eingangs kurz erklärt, die sogenannte Hüllkurve zuständig. Mit Hilfe der Routine SDENVEL lassen sich alle Hüllkurven-Parameter bequem angeben.

Der Aufruf erfolgt mit SYS 40203,TG,A,D,S,R. TG ist dabei die Nummer des anzusprechenden Tongenerators (1-3). Für A, D, S und R sind jeweils Werte zwischen null und 15 erlaubt. Noch einmal kurz zur Wiederholung:

In der Attack-Phase schwillt der Ton von null bis zur maximalen Lautstärke an. Die Zeitdauer dieser Phase wird durch den A-Parameter gesteuert.

In der Decay-Phase, deren Dauer durch den Parameter D beeinflußt wird, fällt die Tonlautstärke auf den durch den Sustain-Parameter S bestimmten sogenannten Haltepegel.

Die Sustain-Phase, in der die Lautstärke auf dem Haltepegel bleibt, ist vom SID zeitlich unbefristet. Ihre Zeitdauer kann beispielsweise durch eine Warteschleife

```
FOR W=1 TO 1000:NEXT W
```

festgelegt werden. Im Gegensatz zu den drei anderen Parametern A, D und R wird durch S also keine Zeitspanne, sondern die

Tonlautstärke nach der Attack-Phase festgelegt. S=15 bedeutet dabei, daß die Lautstärke auf dem Maximalwert der Attack-Phase verbleibt. S=0 hat zur Folge, daß der Ton bereits in der Decay-Phase vollständig ausklingt, seine Lautstärke also auf Null absinkt.

In der Release-Phase, deren Zeitdauer durch den Parameter R festgelegt wird, klingt der Ton vom momentanen Sustain-Pegel auf Null aus. Die den einzelnen Parameterwerten entsprechenden Zeitspannen für A, D und R können Sie der folgenden Tabelle entnehmen:

Parameterwert	Attack (A)	Decay (D)/Release (R)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

SYS 40203,1,9,12,7,14 beispielsweise bewirkt eine Attack-Phase von 250 Millisekunden Dauer, eine 3 Sekunden dauernde Decay-Phase, in der die Lautstärke etwa auf den halben Maximalwert absinkt, und eine mit 15 Sekunden sehr lange dauernde Release-Phase. Der Programmcode von SDENVEL:

```
;*********************************************************
;*                                                       *
;* sdenvel - hüllkurve festlegen                         *
;*                                                       *
;* aufruf: sys 40203,tg,a,d,s,r                          *
;*     tg: stimme (1-3)                                  *
;* a,d,s,r: attack-, decay-, sustain- und                *
;*          release-parameter (siehe text)               *
```

```
1450 ;*                                                        *
1452 ;**********************************************************
1454 ;
1460 sdenvel  jsr $aefd       ;auf komma testen
1470          jsr sdtgnr      ;stimmennr. holen
1480          ldy #5          ;zeiger setzen
1490          jsr sdenv1      ;1. und 2. einstellwert
1500          iny
1510          jsr sdenv1      ;3. und 4. einstellwert
1520          jmp sdumsp      ;sid aktualisieren
1540 sdenv1   jsr sdwtep      ;einstellwert holen
1550          asl             ;*16
1560          asl
1570          asl
1580          asl
1590          sta (zp1),y     ;und setzen
1600          jsr sdwtep      ;einstellwert holen
1610          ora (zp1),y
1620          sta (zp1),y     ;und setzen
1630          rts             ;fertig
1640 ;
1642 ;
```

Wesentlich für den Klang eines Tones ist die eingestellte Wellenform. Beim SID stehen vier Wellenformen zur Auswahl, die für die drei Tongeneratoren getrennt angegeben werden können.

Um eine bestimmte Wellenform einzuschalten, verwenden Sie die Routine SDWAVEON, die mit SYS 40238,TG,WF aufgerufen wird. Ausschalten läßt sich eine Wellenform mit Hilfe der Routine SDWAVEOFF, die durch SYS 40292,TG,WF aufgerufen wird. TG steht wieder für die Nummer des anzusprechenden Tongenerators (1-3). WF ist die Kennziffer der Wellenform:

```
1: Dreieck
2: Sägezahn
3: Rechteck
4: Rauschen
```

Falls Sie als Wellenform "Rechteck" gewählt haben, müssen Sie in PW zusätzlich noch die sogenannte Pulsweite festlegen. Die Pulsweite wird in Prozent angegeben und darf daher Werte von Null bis 100 annehmen. Die Genauigkeit beträgt dabei mehrere Stellen hinter dem Komma. Werte wie 6.11 sind daher durchaus sinnvoll. Bei den anderen Wellenformen hat PW keine Bedeu-

tung. Übrigens kann man auch mehrere Wellenformen gleichzeitig einstellen. Beispielsweise würde

```
SYS 40238,1,1:SYS 40238,1,2
```

für den Tongenerator 1 die Wellenformen "Dreieck" und "Sägezahn" auswählen. In der Regel bringen solche Kombinationen im Hinblick auf den Klang des Tones aber nicht allzu viel. Schon eher von Bedeutung ist da das Umschalten von der einen auf eine andere Wellenform:

```
SYS 40238,2,2:.......:SYS 40292,2,2:SYS 40238,2,3,60
```

Für den Tongenerator 2 wird zunächst die Wellenform "Sägezahn" festgelegt und dann später auf "Rechteck" mit einer Pulsweite von 60% umgeschaltet. Der Programmcode von SDWAVEON und SDWAVEOFF:

```
1644 ;**********************************************************
1646 ;*                                                        *
1648 ;* sdwaveon - wellenform einschalten                      *
1650 ;*                                                        *
1652 ;* aufruf: sys 40238,tg,wf (,pw)                          *
1654 ;*     tg: stimme (1-3)                                   *
1656 ;*     wf: wellenform (1:dreieck, 2:saegezahn,            *
1658 ;*         3:rechteck, 4:rauschen)                        *
1660 ;*     pw: pulsweite (0-100; nur bei wf=3!)               *
1662 ;*                                                        *
1664 ;**********************************************************
1666 ;
1670 sdwaveon jsr sdwvgm      ;gemeinsamer teil
1680          ora sdtab2a,x
1690          sta (zp1),y
1700          cpx #2          ;wellenform 'rechteck'?
1710          bne sdwvft      ;nein, fertig
1720          jsr $0079       ;ja, weitere zeichen?
1730          beq sdwvft      ;nein, fertig
1740          jsr $aefd       ;auf komma testen
1750          jsr $ad8a       ;tastverhaeltnis
1760          lda #<(sdtab3) ;zeiger auf
1770          ldy #>(sdtab3) ;konstante
1780          jsr $ba28       ;mit konstante multiplizieren
1790          jsr $b7f7       ;und nach 16-bit wandeln
1800          lda $15         ;highwert
1810          cmp #16         ;>15?
1820          bcc sdwt1
1830          jmp $b248       ;ja, fehler
1840 sdwt1    ldy #3
```

```
1850            sta (zp1),y    ;wert ablegen
1860            dey
1870            lda $14        ;lowwert
1880            sta (zp1),y
1890 sdwvft     jmp sdumsp     ;sid aktualisieren
1900 ;
1902 ;
1904 ;*****************************************************
1906 ;*                                                   *
1908 ;* sdwaveoff - wellenform ausschalten                *
1910 ;*                                                   *
1912 ;* aufruf: sys 40292,tg,wf                           *
1914 ;*      tg: stimme (1-3)                             *
1916 ;*      wf: wellenform (1-4)                         *
1918 ;*                                                   *
1920 ;*****************************************************
1922 ;
1930 sdwaveoff jsr sdwvgm      ;gemeinsamer teil
1940            and sdtab2b,x
1950            sta (zp1),y
1960            jmp sdumsp     ;sid aktualisieren
1970 ;
2000 ;*** gemeinsamer teil von sdwaveon/off ***************
2020 sdwvgm     jsr $aefd      ;auf komma testen
2030            jsr sdtgnr     ;stimmennr. holen
2040            jsr $aefd      ;auf komma testen
2050            jsr $b79e      ;wellenform
2060            dex            ;-1
2070            cpx #4         ;>3?
2080            bcc sdwv1      ;nein
2090 sderr2     jmp $b248      ;ja, fehler
2100 sdwv1      ldy #4
2110            lda (zp1),y    ;register holen
2120            rts            ;fertig!
2130 ;
2132 ;
```

Nun kommen wir zu den beiden zentralen Routinen des Programmpakets, SDVOICEON und SDVOICEOFF. Die Parameter, die Sie mit den anderen Routinen festgelegt haben, wurden nur in den entsprechenden Registern des SID gespeichert. Zu hören war bisher noch nichts. Das wird sich gleich ändern. Mit Hilfe der Routine SDENVEL haben Sie den Lautstärkeverlauf des Tones, seine Hüllkurve, definiert. Sobald der Tongenerator nun mittels SYS 40328,TG eingeschaltet wird, beginnt die Attack-Phase.

Anschließend geht der Tongenerator in die Decay-Phase und dann in die Sustain-Phase über, in der er auch verbleibt. Erst

wenn Sie durch SYS 40338,TG den Tongenerator wieder ausschalten, wird die Release-Phase eingeleitet.

Zwischen den beiden Aufrufen steht für gewöhnlich eine Warteschleife, mit der sich die Gesamt-Tondauer sehr fein einstellen läßt:

```
100 sys 40328;1:rem tongenerator 1 einschalten
110 for w=1 to 5000:next w:rem warten
120 sys 40338,1:rem tongenerator 1 ausschalten
```

Natürlich können Sie, während der Ton gespielt wird, auch irgendeinen anderen Programmteil abarbeiten lassen. Dann wird es in der Regel allerdings sehr schwierig, eine genaue Tondauer festzulegen. Manchmal möchte man das ja aber auch gar nicht. Hier der Programmcode von SDVOICEON und SDVOICEOFF:

```
2134 ;*****************************************************
2136 ;*                                                   *
2138 ;* sdvoiceon - stimme einschalten                    *
2140 ;*                                                   *
2142 ;* aufruf: sys 40328,tg                              *
2144 ;*     tg: stimme (1-3)                              *
2146 ;*                                                   *
2148 ;*****************************************************
2150 ;
2160 sdvoiceon jsr sdvcrg4a  ;stimmennr. holen
2170           ora #1       ;flag setzen
2180 sdvcrg4b sta (zp1),y
2190           jmp sdumsp   ;sid aktualisieren
2200 ;
2202 ;
2204 ;*****************************************************
2206 ;*                                                   *
2208 ;* sdvoiceoff - stimme ausschalten                   *
2210 ;*                                                   *
2212 ;* aufruf: sys 40338,tg                              *
2214 ;*     tg: stimme (1-3)                              *
2216 ;*                                                   *
2218 ;*****************************************************
2220 ;
2230 sdvoiceoff jsr sdvcrg4a ;stimmennr. holen
2240           and #$fe     ;flag loeschen
2250           jmp sdvcrg4b ;wert zurueckschreiben
2260 ;
2262 ;
```

Eine wesentliche Hilfe, wenn es darum geht, komplexere Klänge zu erzeugen, ist der sogenannte Filter. Der Filter dient dazu, bestimmte Frequenzbereiche eines Tongenerators zu sperren bzw. abzudämpfen.

Ob eine der drei Stimmen über den Filter geschickt werden soll, läßt sich für jede Stimme getrennt festlegen. Allerdings existiert nur ein gemeinsamer Filter für alle Stimmen. Die Routine SDFTON, die durch SYS 40458,TG aufgerufen wird, veranlaßt den SID, die Ausgaben des betreffenden Tongenerators über den Filter zu leiten. Die Routine SDFTOFF, die mit SYS 40488,TG aufgerufen wird, sorgt dafür, daß der Tongenerator wieder ungefiltert zu hören ist.

Übrigens läßt sich der Filter zu jedem beliebigen Zeitpunkt zuschalten. Das erhöht seine Einsatzmöglichkeiten. Bevor Sie mit dem Filter arbeiten können, müssen Sie zunächst einmal die gewünschten Filterparameter definieren. Dazu gibt es die Routine SDFILTER. SDFILTER wird mit SYS 40346,FA,FQ,RS aufgerufen. FA steht für die Filterbetriebsart. Zur Auswahl stehen:

1. Hochpaß: Der Hochpaß läßt nur Frequenzanteile oberhalb der Grenzfrequenz FQ durch.
2. Tiefpaß: Der Tiefpaß läßt nur Frequenzanteile unterhalb der Grenzfrequenz FQ durch.
3. Bandpaß: Der Bandpaß läßt nur die Frequenzanteile in der Umgebung der Mittenfrequenz FQ durch.
4. Bandsperre: Die Bandsperre (auch Notch-Filter genannt) läßt alle Frequenzen, bis auf die Frequenzanteile in der Umgebung der Mittenfrequenz FQ durch.

FQ steht für den Frequenzbereich, in dem der Filter arbeiten soll. Je nach eingestellter Filterbetriebsart hat FQ verschiedene Bedeutungen. Grundsätzlich darf sich FQ im Bereich zwischen 30 und 11.800 Hertz bewegen.

Mit RS wird schließlich noch die sogenannte Resonanz des Filters festgelegt. Ein hoher Resonanzwert (maximal 15) bewirkt eine zusätzliche Verstärkung der Frequenzanteile in der Nähe der Filterfrequenz FQ. Nach soviel Theorie einige Beispiele:

```
SYS 40346,1,500,12
```

stellt die Betriebsart "Hochpaß" ein, wobei die Filterfrequenz 500 Hertz beträgt. Der Filter läßt nun also nur noch Frequenzen oberhalb von 500 Hertz durch. Frequenzen in der Nähe von 500 Hertz werden wegen des Resonanzwertes 12 besonders verstärkt.

```
SYS 40346,2,2000,0
```

stellt die Betriebsart "Tiefpaß" ein, wobei die Grenzfrequenz 2.000 Hertz beträgt. Frequenzen oberhalb von 2.000 Hertz werden also nicht mehr durchgelassen. Wegen des Resonanzwertes 0 erfolgt keine zusätzliche Verstärkung von Frequenzanteilen.

```
SYS 40346,4,8000,4
```

stellt die Filterbetriebsart "Bandsperre" ein. Alle Frequenzen außer derjenigen in der Umgebung von 8.000 Hertz werden durchgelassen. Frequenzen knapp unterhalb und knapp oberhalb von 8.000 Hertz werden wegen RS=4 leicht verstärkt. Der Programmcode von SDFILTER, SDFTON und SDFTOFF:

```
2264 ;*****************************************************
2266 ;*                                                   *
2268 ;* sdfilter - filter einstellen                      *
2270 ;*                                                   *
2272 ;* aufruf: sys 40346,fa,fq,rs                        *
2274 ;*      fa: filterbetriebsart (1-4; siehe text)      *
2276 ;*      fq: grenzfrequenz (30-11800 hz)              *
2278 ;*      rs: resonanz (0-15)                          *
2280 ;*                                                   *
2282 ;*****************************************************
2284 ;
2290 sdfilter jsr $aefd      ;auf komma testen
2300          jsr $b79e      ;filterart holen
2310          dex            ;-1
2320          cpx #4         ;>3?
2330          bcc sdfl1      ;nein, weiter
2340 sdflerr  jmp $b248      ;ja, fehler
2350 sdfl1    lda sdzwsp+24  ;filter-register
2360          and #143       ;flags loeschen
2370          ora sdtab6,x   ;und entspr. setzen
2380          sta sdzwsp+24
2390          jsr $aefd      ;auf komma testen
2400          jsr $ad8a      ;filterfrequenz holen
2410          lda #<(sdtab7) ;zeiger auf konstante
2420          ldy #>(sdtab7) ;(grenzfrequenz 11800 hz)
```

```
2430          jsr $bc5b      ;vergleich
2440          bpl sdfl2      ;positiv
2450          beq sdfl2      ;oder null, dann weiter
2460          jmp $b248      ;sonst fehler
2470 sdfl2    lda #<(sdtab8) ;zeiger auf
2480          ldy #>(sdtab8) ;konstante
2490          jsr $b867      ;addition
2500          lda #<(sdtab9) ;zeiger auf
2510          ldy #>(sdtab9) ;konstante
2520          jsr $ba28      ;multiplikation
2530          jsr $b7f7      ;nach 16-bit wandeln
2540          lda $15        ;highwert
2550          cmp #8         ;>7?
2560          bcs sdflerr    ;ja, fehler
2570          lda $14        ;lowwert
2580          and #7
2590          sta sdzwsp+21  ;in register ablegen
2600          lda $14
2610          lsr
2620          lsr
2630          lsr
2640          sta sdzwsp+22
2650          lda $15        ;highwert
2660          asl
2670          asl
2680          asl
2690          asl
2700          asl
2710          ora sdzwsp+22
2720          sta sdzwsp+22
2730          jsr sdwtep     ;resonanz holen
2740          asl
2750          asl
2760          asl
2770          asl
2780          ora sdzwsp+23
2790          sta sdzwsp+23  ;in register ablegen
2800          jmp sdumsp     ;sid aktualisieren
2810 ;
2812 ;
2814 ;*********************************************************
2816 ;*                                                       *
2818 ;* sdfton - filter zuschalten                            *
2820 ;*                                                       *
2822 ;* aufruf: sys 40458,tg                                  *
2824 ;*      tg: stimme (1-3)                                 *
2826 ;*                                                       *
2828 ;*********************************************************
2830 ;
2840 sdfton   jsr sdvcftgm   ;gemeinsamer teil
2850          ora sdtab5a,x  ;stimmen-Flag setzen
2860 sdvcfnjp sta sdzwsp+23
2870          jmp sdumsp     ;sid aktualisieren
```

```
2880 ;
2890 ;
2910 ;*** gemeinsamer teil von sdfton/off *****************
2930 sdvcftgm jsr $aefd      ;auf komma testen
2940          jsr $b79e      ;stimmennr. holen
2950          dex            ;-1
2960          cpx #4         ;>3?
2970          bcc sdvg1      ;nein, weiter
2980          jmp $b248      ;ja, fehler
2990 sdvg1    lda sdzwsp+23  ;entspr. filterreg. holen
3000          rts            ;fertig!
3010 ;
3012 ;
3014 ;*****************************************************
3016 ;*                                                   *
3018 ;* sdftoff - filter abschalten                       *
3020 ;*                                                   *
3022 ;* aufruf: sys 40488,tg                              *
3024 ;*     tg: stimme (1-3)                              *
3026 ;*                                                   *
3028 ;*****************************************************
3030 ;
3040 sdftoff  jsr sdvcftgm   ;gemeinsamer teil
3050          and sdtab5b,x  ;stimmen-flag loeschen
3060          jmp sdvcfnjp   ;zur sdfton-routine
3070 ;
3072 ;
```

Der SID erlaubt es auch, eine Stimme durch eine andere Stimme synchronisieren zu lassen. Dazu muß nur das entsprechende Bit im Kontroll-Register gesetzt werden.

Dazu dient die Routine SDSYNON. Mit SDSYNOFF läßt sich die Synchronisation wieder abbrechen. Beide Routinen benötigen als Parameter nur die Nummer TG der Stimme, die synchronisiert werden soll: SDSYNON wird mit SYS 40497,TG aufgerufen, SDSYNOFF durch SYS 40505,TG.

Welche Stimme durch welche Stimme synchronisiert werden kann, ist fest vorgegeben:

TG=1: Stimme 1 wird durch Stimme 3 synchronisiert.
TG=2: Stimme 2 wird durch Stimme 1 synchronisiert.
TG=3: Stimme 3 wird durch Stimme 2 synchronisiert.

Hier der Programmcode:

```
3074 ;*******************************************************
3076 ;*                                                     *
3078 ;* sdsynon - synchronisation einschalten               *
3080 ;*                                                     *
3082 ;* aufruf: sys 40497,tg                                *
3084 ;*     tg: stimme (1-3)                                *
3086 ;*                                                     *
3088 ;*******************************************************
3090 ;
3100 sdsynon  jsr sdvcrg4a  ;stimmennr. holen
3110          ora #02       ;flag setzen
3120          jmp sdvcrg4b  ;wert zurueckschreiben
3130 ;
3132 ;
3134 ;*******************************************************
3136 ;*                                                     *
3138 ;* sdsynoff - synchronisation ausschalten              *
3140 ;*                                                     *
3142 ;* aufruf: sys 40505,tg                                *
3144 ;*     tg: stimme (1-3)                                *
3146 ;*                                                     *
3148 ;*******************************************************
3150 ;
3160 sdsynoff jsr sdvcrg4a  ;stimmennr. holen
3170          and #$fd      ;flag loeschen
3180          jmp sdvcrg4b  ;wert zurueckschreiben
3190 ;
3192 ;
```

Eine weitere Besonderheit des SID ist die Erzeugung von sogenannten Ringmodulator-Produkten aus zwei Stimmen. Ähnlich wie bei der Synchronisation werden dabei zwei Stimmen zu einer "neuen" Stimme überlagert. Dazu existiert ebenfalls ein entsprechendes Bit im Kontroll-Register der Tongeneratoren, das mit SDRINGON gesetzt werden kann. SDRINGOFF beendet die Ringmodulation.

Beide Routinen benötigen als Parameter nur die Nummer TG der Stimme, die ringmoduliert werden soll: SDRINGON wird mit SYS 40513,TG aufgerufen, SDRINGOFF durch SYS 40521,TG. Welche Stimme durch welche Stimme ringmoduliert werden kann, ist fest vorgegeben:

TG=1: Ringmodulator-Produkt aus den Stimmen 1 und 3.
TG=2: Ringmodulator-Produkt aus den Stimmen 2 und 1.
TG=3: Ringmodulator-Produkt aus den Stimmen 3 und 2.

Zu beachten ist, daß die Wellenform der betreffenden Stimme auf "Dreieck" eingestellt sein muß. Ansonsten bleibt die Ringmodulation unhörbar! Hier der Programmcode:

```
3194 ;*****************************************************
3196 ;*                                                   *
3198 ;* sdringon - ringmodulation einschalten             *
3200 ;*                                                   *
3202 ;* aufruf: sys 40513,tg                              *
3204 ;*     tg: stimme (1-3)                              *
3206 ;*                                                   *
3208 ;*****************************************************
3210 ;
3220 sdringon jsr sdvcrg4a    ;stimmennr. holen
3230          ora #04         ;flag setzen
3240          jmp sdvcrg4b    ;wert zurueckschreiben
3250 ;
3252 ;
3254 ;*****************************************************
3256 ;*                                                   *
3258 ;* sdringoff - ringmodulation ausschalten            *
3260 ;*                                                   *
3262 ;* aufruf: sys 40521,tg                              *
3264 ;*     tg: stimme (1-3)                              *
3266 ;*                                                   *
3268 ;*****************************************************
3270 ;
3280 sdringoff jsr sdvcrg4a   ;stimmennr. holen
3290          and #$fb        ;flag loeschen
3300          jmp sdvcrg4b    ;wert zurueckschreiben
```

Damit haben wir alle Routinen durch. Bitte vergessen Sie nun nicht, noch die nachfolgenden gemeinsamen Unterprogramme und die Tabellen abzutippen:

```
3310 ;
3320 ;
3330 ;
3340 ;
3350 ;
3390 ;*** gemeinsame unterprogramme **********************
3400 ;
3410 ;
3430 ;*** aktualisierung der sid-register ***
3450 sdumsp   ldx #24        ;anzahl der register-1
3460 sdumsplp lda sdzwsp,x   ;zwischenspeicher in sid
```

```
3470            sta 54272,x     ;umspeichern
3480            dex
3490            bpl sdumsplp    ;alle register
3500            rts             ;fertig!
3510 ;
3520 ;
3540 ;*** wert zwischen 0 und 15 holen und testen ***
3560 sdwtfq     jsr $b79e       ;wert holen
3570            cpx #16         ;>15?
3580            bcc sdwq1       ;nein
3590 sderr1     jmp $b248       ;ja, 'illegal quantity'
3600 sdwq1      rts             ;fertig!
3610 ;
3620 ;
3640 ;*** wert zwischen 0 und 15 holen ***
3660 sdwtep     sty zw          ;zeiger zwischenspeichern
3670            jsr $aefd       ;auf komma testen
3680            jsr sdwtfq      ;wert holen und testen
3690            ldy zw          ;zeiger
3700            txa             ;wert in akku
3710            rts             ;fertig!
3720 ;
3730 ;
3750 ;*** nummer der stimme einlesen und testen ***
3770 sdtgnr     jsr $b79e       ;nummer holen
3780            dex             ;-1
3790            cpx #3          ;>2?
3800            bcs sderr1      ;nein, fehler
3810            lda sdtab1a,x   ;register-adr. fuer stimme
3820            sta zp1
3830            lda sdtab1b,x   ;highwert
3840            sta zp2
3850            rts             ;fertig!
3860 ;
3870 ;
3890 ;** bearbeitung der stimm-Register 4 (kontrollreg.) ***
3910 sdvcrg4a jsr $aefd         ;auf komma testen
3920            jsr sdtgnr      ;stimmennr. holen
3930            ldy #4          ;zeiger
3940            lda (zp1),y     ;register
3950            rts             ;fertig!
3960 ;
3970 ;
3980 ;
3990 ;
4000 ;
4040 ;*** sid-tabellen ***********************************
4050 ;
4060 ;*** low-werte der register-adressen ***
4070 sdtab1a    .by <(sdzwsp),<(sdzwsp+7),<(sdzwsp+14)
4080 sdtab1b    .by >(sdzwsp),>(sdzwsp+7),>(sdzwsp+14)
4090 ;
4110 ;*** tabellen fuer wellenform-flags ***
```

```
4120 sdtab2a  .by 16,32,64,128
4130 sdtab2b  .by 239,223,191,127
4140 ;
4160 ;*** konstante fuer tastverhaeltnisberechnung ***
4170 sdtab3   .by $86,$23,$cc,$cc,$cd
4180 ;
4200 ;*** konstante fuer frequenzberechnung ***
4210 sdtab4   .by $85,$08,$3a,$33,$be
4220 ;
4240 ;*** tabellen fuer filter-flags ***
4250 sdtab5a  .by 1,2,4,8
4260 sdtab5b  .by 254,253,251,247
4270 ;
4290 ;*** tabelle fuer filterarten ***
4300 sdtab6   .by 64,16,32,80
4310 ;
4330 ;*** konstanten fuer filterfrequenzberechnung ***
4340 sdtab7   .by $85,$70,$00,$00,$00
4350 sdtab8   .by $85,$f0,$00,$00,$00
4360 sdtab9   .by $7e,$30,$8d,$3d,$96
4370 ;
4390 ;*** tabellen fuer notenumrechnung ***
4400 sdtab10  .by $43,35,$44,35,$45,$46,35,$47,35,$41,35,$48
4410 sdtab11  .by $38,$8b,$81,$93,$45,$9c,$90,$a5,$68,$af,$d6,$b9
4420          .by $e4,$c4,$99,$d0,$ff,$dc,$24,$ea,$10,$f8,$cf,$06
4430 ;
4435 ;
4440 ;
4445 ;
4450 ;
4460 ;*** zwischenspeicher fuer sid-register *************
4480 sdzwsp   .by 0,0,0,0,0,0,0,0,0,0
4490          .by 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

BASIC-Lader für alle Maschinensprache-Routinen

Der nachfolgende BASIC-Lader enthält sämtliche in diesem Abschnitt vorgestellten Maschinensprache-Routinen zur Sound-Programmierung. Zudem habe ich in den Kommentarzeilen noch einmal die Routinen mit ihren Aufrufparametern aufgelistet. Diese REM-Zeilen müssen Sie natürlich nicht mit abtippen. Falls Sie mit den Routinen häufiger arbeiten möchten, wird es das Beste sein, wenn Sie sich diese Auflistung fotokopieren.

```
100 rem ****************************************************
102 rem *                                                  *
104 rem * programm: sound                                  *
106 rem * programmlaenge: 706 bytes                        *
108 rem * routinen zur soundprogrammierung                 *
110 rem *                                                  *
```

```
112 rem ***************************************************
114 rem *                                                 *
116 rem * sdclear - sid-register loeschen                 *
118 rem *                                                 *
120 rem * aufruf: sys 40000                               *
122 rem *                                                 *
124 rem ***************************************************
126 rem *                                                 *
128 rem * sdvolume - gesamtlautstaerke einstellen         *
130 rem *                                                 *
132 rem * aufruf: sys 40013,ls                            *
134 rem *     ls: lautstaerke (0-15)                      *
136 rem *                                                 *
138 rem ***************************************************
140 rem *                                                 *
142 rem * sdfreq - frequenz einer stimme einstellen       *
144 rem *                                                 *
146 rem * aufruf: sys 40032,tg,fr                         *
148 rem *     tg: stimme (1-3)                            *
150 rem *     fr: frequenz in Hertz (0-3848)              *
152 rem *                                                 *
154 rem ***************************************************
156 rem *                                                 *
158 rem * sdnote - musiknote eingeben                     *
160 rem *                                                 *
162 rem * aufruf: sys 40068,tg,nt$                        *
164 rem *     tg: stimme (1-3)                            *
166 rem *    nt$: musiknote ('c0'-'a#7')                  *
168 rem *                                                 *
170 rem ***************************************************
172 rem *                                                 *
174 rem * sdenvel - hüllkurve festlegen                   *
176 rem *                                                 *
178 rem * aufruf: sys 40203,tg,a,d,s,r                    *
180 rem *     tg: stimme (1-3)                            *
182 rem * a,d,s,r: attack-, decay-, sustain- und          *
184 rem *         release-parameter (siehe text)          *
186 rem *                                                 *
188 rem****************************************************
190 rem *                                                 *
192 rem * sdwaveon - wellenform einschalten               *
194 rem *                                                 *
196 rem * aufruf: sys 40238,tg,wf (,pw)                   *
198 rem *     tg: stimme (1-3)                            *
200 rem *     wf: wellenform (1:dreieck, 2:saegezahn,     *
202 rem *         3:rechteck, 4:rauschen)                 *
204 rem *     pw: pulsweite (0-100; nur bei wf=3!)        *
206 rem *                                                 *
208 rem****************************************************
210 rem *                                                 *
212 rem * sdwaveoff - wellenform ausschalten              *
214 rem *                                                 *
216 rem * aufruf: sys 40292,tg,wf                         *
```

```
218 rem *     tg: stimme (1-3)                                *
220 rem *     wf: wellenform (1-4)                            *
222 rem *                                                     *
224 rem ***************************************************
226 rem *                                                     *
228 rem * sdvoiceon - stimme einschalten                      *
230 rem *                                                     *
232 rem * aufruf: sys 40328,tg                                *
234 rem *     tg: stimme (1-3)                                *
236 rem *                                                     *
238 rem ***************************************************
240 rem *                                                     *
242 rem * sdvoiceoff - stimme ausschalten                     *
244 rem *                                                     *
246 rem * aufruf: sys 40338,tg                                *
248 rem *     tg: stimme (1-3)                                *
250 rem *                                                     *
252 rem ***************************************************
254 rem *                                                     *
256 rem * sdfilter - filter einstellen                        *
258 rem *                                                     *
260 rem * aufruf: sys 40346,fa,fq,rs                          *
262 rem *     fa: filterbetriebsart (1-4; siehe text)         *
264 rem *     fq: grenzfrequenz (30-11800 hz)                 *
266 rem *     rs: resonanz (0-15)                             *
268 rem *                                                     *
270 rem ***************************************************
272 rem *                                                     *
274 rem * sdfton - filter zuschalten                          *
276 rem *                                                     *
278 rem * aufruf: sys 40458,tg                                *
280 rem *     tg: stimme (1-3)                                *
282 rem *                                                     *
284 rem ***************************************************
286 rem *                                                     *
288 rem * sdftoff - filter abschalten                         *
290 rem *                                                     *
292 rem * aufruf: sys 40488,tg                                *
294 rem *     tg: stimme (1-3)                                *
296 rem *                                                     *
298 rem ***************************************************
300 rem *                                                     *
302 rem * sdsynon - synchronisation einschalten               *
304 rem *                                                     *
306 rem * aufruf: sys 40497,tg                                *
308 rem *     tg: stimme (1-3)                                *
310 rem *                                                     *
312 rem ***************************************************
314 rem *                                                     *
316 rem * sdsynoff - synchronisation ausschalten              *
318 rem *                                                     *
320 rem * aufruf: sys 40505,tg                                *
322 rem *     tg: stimme (1-3)                                *
```

```
324 rem *                                                   *
326 rem *****************************************************
328 rem *                                                   *
330 rem * sdringon - ringmodulation einschalten             *
332 rem *                                                   *
334 rem * aufruf: sys 40513,tg                              *
336 rem *      tg: stimme (1-3)                             *
338 rem *                                                   *
340 rem *****************************************************
342 rem *                                                   *
344 rem * sdringoff - ringmodulation ausschalten            *
346 rem *                                                   *
348 rem * aufruf: sys 40521,tg                              *
350 rem *      tg: stimme (1-3)                             *
352 rem *                                                   *
354 rem *****************************************************
356 :
358 :
360 rem basic-ende auf 39999 herabsetzen
370 poke 55,63:poke 56,156:clr
400 rem *** datas einlesen ***
410 restore:zl=1000
420 ad=40000:rem startadresse der routine
430 ps=0:z=0
440 read wt:if wt=-1 then print "datas ok!":end
450 poke ad,wt:ad=ad+1:z=z+1
460 ps=ps+wt:rem pruefsumme
470 if z<10 then 440:rem eine zeile lesen
480 read wt
485 if ps<>wt then print "data-fehler in zeile";zl;"!":end
490 zl=zl+10:goto 430
500 :
510 :
990 rem *** datas ***
1000 data 162,24,169,0,157,233,158,202,16,250,1371
1010 data 76,81,158,32,253,174,32,93,158,173,1230
1020 data 1,159,41,240,5,2,141,1,159,76,825
1030 data 81,158,32,253,174,32,116,158,32,253,1289
1040 data 174,32,138,173,169,165,160,158,32,40,1241
1050 data 186,32,247,183,160,0,165,20,145,251,1389
1060 data 200,165,21,145,251,76,81,158,32,253,1382
1070 data 174,32,116,158,32,253,174,32,158,173,1302
1080 data 32,163,182,160,1,177,34,133,2,136,1020
1090 data 177,34,133,253,200,162,0,221,197,158,1535
1100 data 240,8,232,224,12,144,246,76,8,175,1365
1110 data 165,2,201,35,208,11,232,221,197,158,1430
1120 data 208,241,200,177,34,133,2,138,10,170,1313
1130 data 189,209,158,133,253,232,189,209,158,133,1863
1140 data 254,165,2,56,233,48,201,8,144,3,1114
1150 data 76,72,178,73,7,208,11,160,0,177,962
1160 data 34,201,72,240,241,76,253,156,170,160,1603
1170 data 0,177,34,201,72,208,5,56,102,254,1109
1180 data 48,2,70,254,102,253,202,208,249,160,1548
```

```
1190 data 0,165,253,145,251,200,165,254,145,251,1829
1200 data 76,81,158,32,253,174,32,116,158,160,1240
1210 data 5,32,29,157,200,32,29,157,76,81,798
1220 data 158,32,104,158,10,10,10,10,145,251,888
1230 data 32,104,158,17,251,145,251,96,32,111,1197
1240 data 157,29,152,158,145,251,224,2,208,39,1365
1250 data 32,121,0,240,34,32,253,174,32,138,1056
1260 data 173,169,160,160,158,32,40,186,32,247,1357
1270 data 183,165,21,201,16,144,3,76,72,178,1059
1280 data 160,3,145,251,136,165,20,145,251,76,1352
1290 data 81,158,32,111,157,61,156,158,145,251,1310
1300 data 76,81,158,32,253,174,32,116,158,32,1112
1310 data 253,174,32,158,183,202,224,4,144,3,1377
1320 data 76,72,178,160,4,177,251,96,32,135,1181
1330 data 158,9,1,145,251,76,81,158,32,135,1046
1340 data 158,41,254,76,141,157,32,253,174,32,1318
1350 data 158,183,202,224,4,144,3,76,72,178,1244
1360 data 173,1,159,41,143,29,178,158,141,1,1024
1370 data 159,32,253,174,32,138,173,169,182,160,1472
1380 data 158,32,91,188,16,5,240,3,76,72,881
1390 data 178,169,187,160,158,32,103,184,169,192,1532
1400 data 160,158,32,40,186,32,247,183,165,21,1224
1410 data 201,8,176,199,165,20,41,7,141,254,1212
1420 data 158,165,20,74,74,74,141,255,158,165,1284
1430 data 21,10,10,10,10,10,13,255,158,141,638
1440 data 255,158,32,104,158,10,10,10,10,13,760
1450 data 0,159,141,0,159,76,81,158,32,22,828
1460 data 158,29,170,158,141,0,159,76,81,158,1130
1470 data 32,253,174,32,158,183,202,224,4,144,1406
1480 data 3,76,72,178,173,0,159,96,32,22,811
1490 data 158,61,174,158,76,16,158,32,135,158,1126
1500 data 9,2,76,141,157,32,135,158,41,253,1004
1510 data 76,141,157,32,135,158,9,4,76,141,929
1520 data 157,32,135,158,41,251,76,141,157,162,1310
1530 data 24,189,233,158,157,0,212,202,16,247,1438
1540 data 96,32,158,183,224,16,144,3,76,72,1004
1550 data 178,96,132,2,32,253,174,32,93,158,1150
1560 data 164,2,138,96,32,158,183,202,224,3,1202
1570 data 176,232,189,146,158,133,251,189,149,158,1781
1580 data 133,252,96,32,253,174,32,116,158,160,1406
1590 data 4,177,251,96,233,240,247,158,158,158,1722
1600 data 16,32,64,128,239,223,191,127,134,35,1189
1610 data 204,204,205,133,8,58,51,190,1,2,1056
1620 data 4,8,254,253,251,247,64,16,32,80,1209
1630 data 133,112,0,0,0,133,240,0,0,0,618
1640 data 126,48,141,61,150,67,35,68,35,69,800
1650 data 70,35,71,35,65,35,72,56,139,129,707
1660 data 147,69,156,144,165,104,175,214,185,228,1587
1670 data 196,153,208,255,220,36,234,16,248,207,1773
1680 data 6,0,0,0,0,0,0,0,0,0,6
1690 data 0,0,0,0,0,0,0,0,0,0,0
1700 data 0,0,0,0,0,0,0,0,0,0,0
2000 data -1:rem endmarkierung
```

Einstellbeispiele

Nachdem wir nun alles an Routinen beieinander haben, was wir für die Sound-Programmierung benötigen, wollen wir einmal ein wenig experimentieren. Für welche Art des Abtippens (Assembler oder BASIC) auch immer Sie sich entschieden haben, für die nun folgenden Programme muß sich das Programmpaket im Rechner befinden. Ansonsten bekommen Sie sofort einen Absturz!

Am einfachsten ist es natürlich mit dem BASIC-Lader. Einfach den BASIC-Lader starten und anschließend mit NEW löschen. Die Routinen sind nun gebrauchsfertig installiert und der BASIC-Speicher ist frei für Ihre ersten Sound-Programme. Eines gleich vornweg: Die Einstellung aller SID-Parameter ist in den meisten Fällen sehr subjektiv. Insbesondere, wenn es darum geht, Musikinstrumente nachzuahmen, können die Geschmäcker sehr differieren. Was für den einen wie eine Trompete klingt, erinnert den anderen vielleicht eher an eine Flöte.

Die hier vorgestellten Einstellbeispiele sollen Ihnen deshalb nur eine Orientierung geben, wie sich in etwa ein bestimmtes Geräusch oder eine bestimmte Imitation erreichen läßt. Beginnen wir mit etwas ganz Einfachem, einem Signalton, der vom Tongenerator 1 erzeugt werden soll:

```
100 rem programm: signalton
110 sys 40000:rem sid initialisieren
120 sys 40013,15:rem volle lautstaerke
130 sys 40032,1,500:rem frequenz 500 hertz
140 sys 40238,1,1:rem wellenform dreieck
150 sys 40203,1,0,0,15,0:rem huellkurve
160 :
170 sys 40328,1:rem stimme ein
180 for w=1 to 5000:next w:rem warteschleife
190 sys 40338,1:rem stimme aus
200 :
210 end
```

Indem Sie die Warteschleife in Programmzeile 180 entsprechend variieren, können Sie einen Signalton von beliebiger Dauer spielen lassen. Schon etwas komplexer ist das Erzeugen eines Schußgeräusches:

```
100 rem programm: schussgeraeusch
110 sys 40000:rem sid initialisieren
120 sys 40013,15:rem volle lautstaerke
130 sys 40032,1,3250:rem frequenz 3250 hertz
140 sys 40238,1,4:rem wellenform rauschen
150 sys 40203,1,0,9,0,0:rem huellkurve
160 :
170 sys 40328,1:rem stimme ein
180 for w=1 to 1000:next w:rem warteschleife
190 sys 40338,1:rem stimme aus
200 :
210 end
```

Um ein Explosionsgeräusch erzeugen zu können, benötigen wir den Filter:

```
100 rem programm: explosionsgeraeusch
110 sys 40000:rem sid initialisieren
120 sys 40013,15:rem volle lautstaerke
130 sys 40032,1,500:rem frequenz 500 hertz
140 sys 40238,1,4:rem wellenform rauschen
150 sys 40203,1,0,11,3,12:rem huellkurve
160 rem filter hochpass, frequenz 500 hertz, resonanz 12
170 sys 40346,1,500,12
180 sys 40458,1:rem stimme 1 ueber filter leiten
190 :
200 sys 40328,1:rem stimme ein
210 for w=1 to 2500:next w:rem warteschleife
220 sys 40338,1:rem stimme aus
230 :
240 end
```

Die beiden folgenden Beispiele sollen Ihnen zeigen, welche komplexen Klänge sich mit Hilfe der Synchronisation und der Ringmodulation von Stimmen erreichen lassen. Beide Programme erzeugen sirenenartige Geräusche. Lassen Sie sich überraschen:

```
100 rem programm: synchronisation
110 sys 40000:rem sid initialisieren
120 sys 40013,15:rem volle lautstaerke
130 sys 40203,1,0,0,15,0:rem huellkurve stimme 1
140 sys 40238,1,3,40:rem wellenform (stimme1) rechteck
150 sys 40238,3,2:rem wellenform (stimme3) saegezahn
160 sys 40497,1:rem synchronisation einschalten
170 sys 40032,3,150:rem frequenz (stimme3) 150 hertz
180 :
190 sys 40328,1:sys 40328,3:rem beide stimmen ein
200 for z=100 to 2000 step 3
210 sys 40032,1,z:rem frequenz schrittweise aendern
220 next z
```

```
230 sys 40000:rem sid initialisieren
240 :
250 end
100 rem programm: ringmodulation
110 sys 40000:rem sid initialisieren
120 sys 40013,15:rem volle lautstaerke
130 sys 40203,1,0,0,15,0:rem huellkurve stimme 1
140 sys 40203,1,0,0,15,0:rem huellkurve stimme 2
150 sys 40238,1,1,40:rem wellenform (stimme1) dreieck
160 sys 40238,3,3,40:rem wellenform (stimme3) rechteck
170 sys 40513,1:rem ringmodulation einschalten
180 :
190 sys 40328,1:sys 40328,3:rem beide stimmen ein
200 :
210 rem frequenzen schrittweise aendern
220 for z1=1 to 2000 step 100
230 for z2=1 to 2000 step 50
240 sys 40032,3,z2:rem frequenz stimme3
250 next z2
260 sys 40032,1,z1:rem frequenz stimme1
270 next z1
280 sys 40000:rem sid initialisieren
290 :
300 end
```

Zum Abschluß und als Überleitung zum nächsten Abschnitt soll es etwas musikalisch werden. Die nächsten drei Beispiele ahmen den Ton einer Flöte, einer Oboe sowie eines Banjos nach:

```
100 rem programm: floete
110 sys 40000:rem sid initialisieren
120 sys 40013,15:rem volle lautstaerke
130 sys 40032,1,600:rem frequenz 600 hertz
140 sys 40238,1,1:rem wellenform dreieck
150 sys 40203,1,8,5,15,8:rem huellkurve
160 :
170 sys 40328,1:rem stimme ein
180 get eg$:if eg$="" then 180:rem auf tastendruck warten
190 sys 40338,1:rem stimme aus
200 :
210 end
100 rem programm: oboe
110 sys 40000:rem sid initialisieren
120 sys 40013,15:rem volle lautstaerke
130 sys 40032,1,450:rem frequenz 450 hertz
140 sys 40238,1,3,6.11:rem wellenform rechteck, pulsweite 6.11%
150 sys 40203,1,4,9,15,8:rem huellkurve
160 :
170 sys 40328,1:rem stimme ein
180 get eg$:if eg$="" then 180:rem auf tastendruck warten
190 sys 40338,1:rem stimme aus
```

```
200 :
210 end
100 rem programm: banjo
110 sys 40000:rem sid initialisieren
120 sys 40013,15:rem volle lautstaerke
130 sys 40032,1,450:rem frequenz 450 hertz
140 sys 40238,1,2:rem wellenform saegezahn
150 sys 40203,1,0,9,0,0:rem huellkurve
160 rem filter notch, frequenz 2000 hertz, resonanz 15
170 sys 40346,4,2000,15
180 sys 40458,1:rem stimme 1 ueber filter leiten
190 :
200 sys 40328,1:rem stimme ein
210 for w=1 to 2000:next w:rem warteschleife
220 sys 40338,1:rem stimme aus
230 :
240 end
```

Diese Einstellbeispiele sollen genügen. Wie Sie gesehen (oder besser: gehört) haben, lassen sich dem SID die interessantesten Geräusche und Töne entlocken.

7.4 Etwas Musiktheorie

Wie Sie wahrscheinlich wissen, ist die sogenannte Tonleiter in der Musik in einzelne Oktaven und diese wiederum in verschiedene Noten unterteilt. Eine Musiknote ist also durch ihre Notenbezeichnung und ihre Oktave charakterisiert. Jeder Musiknote entspricht eine bestimmte Frequenz. Den Noten der 1. Oktave beispielsweise sind die folgenden Frequenzen zugeordnet:

```
C1  32.7 Hz
C#1 34.6 Hz
D1  36.7 Hz
D#1 38.9 Hz
E1  41.2 Hz
F1  43.7 Hz
F#1 46.2 Hz
G1  49.0 Hz
G#1 51.9 Hz
A1  55.0 Hz
A#1 58.3 Hz
H1  61.7 Hz
```

In dieser Tabelle sind gleichzeitig auch alle Notenbezeichnungen aufgelistet. Die Frequenzen der jeweils nächsten Oktave lassen

sich leicht berechnen. Der Frequenzwert wird einfach verdoppelt. Der Note "C2" entspricht also die Frequenz 65.4 Hertz. Wie Sie schon wissen, sind die Tongeneratoren des SID in der Lage, Frequenzen zwischen 0 und 3.848 Hertz zu erzeugen. Die SID-Tonleiter reicht daher von "C0" (16,4 Hz) bis "A#7" (3729,3 Hz). Insgesamt stehen also acht Oktaven zur Verfügung.

In der Regel wird man mit zwei bis drei Oktaven auskommen. Musikstücke, die sich über mehr als drei Oktaven erstrecken sind sehr selten. Außerdem empfindet das menschliche Ohr Töne unterhalb etwa 100 Hertz (Note "G2") nur noch als dumpfes Brummen. Und auch die Noten der höchsten Oktave klingen meist nur noch wie ein mehr oder minder schrilles Pfeifen, je nachdem, welche Wellenform und Hüllkurve man gewählt hat.

Für alle Fälle finden Sie aber im Anhang eine Tabelle mit allen Noten und ihren zugehörigen Frequenzen.

7.5 Musik mit dem Commodore 64

Da es natürlich recht umständlich ist, sich immer wieder die zu den Noten gehörenden Frequenzen herauszusuchen, enthält das im vorherigen Abschnitt vorgestellte Maschinensprachepaket eine spezielle Routine SDNOTE, bei der man die Noten in der gewohnten Schreibweise angeben kann. Kurz zur Wiederholung: SDNOTE wird mit

```
SYS 40068,TG,NT$
```

aufgerufen. TG steht für die Nummer des Tongenerators NT$ enthält die Note. Da der Commodore 64 natürlich nicht über die Klangqualitäten eines tatsächlichen Musikinstruments, etwa eines Klaviers, verfügt, muß man manchmal etwas "tricksen", um wenigstens einen einigermaßen guten Klang zu bekommen. Der Trick besteht nun darin, die Frequenz einer Note zu variieren, d.h. die zugehörige Frequenz etwas zu erhöhen oder zu verringern. Ein "A" in der dritten Oktave beispielsweise läßt man dann nicht mehr mit 220,0 Hertz spielen, wie es ja eigentlich korrekt wäre, sondern vielleicht mit 218,5 oder 222,0 Hertz.

In diesem Fall können Sie die Routine SDNOTE natürlich nicht mehr verwenden und müssen sich mit der direkten Frequenzeinstellung über SDFREQ begnügen. Noch einmal zur Erinnerung, SDFREQ wird so aufgerufen:

```
SYS 40032,TG,FR
```

Ob eine solche "Verfälschung" der Noten erforderlich ist, hängt entscheidend von der Wahl des zu imitierenden Musikinstruments ab und nicht zuletzt natürlich auch von Ihrem "musikalischen" Gehör. Im folgenden möchte ich einmal davon ausgehen, daß Sie nicht allzu empfindlich sind, und werde deshalb SDNOTE verwenden.

Neben den Noten selbst muß auch ihre jeweilige Spieldauer festgelegt werden. Auch hier hat der Commodore 64 einen Nachteil oder auch Vorteil, je nachdem, wie man es sieht. Um die Notendauer einzustellen, muß man sich einer sogenannten "Warteschleife" bedienen, beispielsweise:

```
FOR W=1 TO 100:NEXT W
```

Diese Warteschleife hat den Nachteil, daß man die Spieldauer einer Note nicht exakt einstellen kann. Die Abarbeitung ein und derselben Schleife kann nämlich durchaus unterschiedlich lange dauern, je nachdem, wo im Programm (weiter vorne oder weiter hinten) sie steht.

Auf der anderen Seite läßt sich die Spieldauer durch die Warteschleife praktisch völlig frei gestalten. Man ist nicht auf die in der Musik üblichen "ganzen", "halben" oder "viertel" Noten beschränkt. Im Hinblick auf einen besseren Klang kann das sehr nützlich sein. Möchten Sie sich trotzdem an die übliche Notendauer halten, so gilt die folgende Regel:

```
FOR W=1 TO 1000:NEXT W
```

entspricht in etwa einer ganzen Note,

```
FOR W=1 TO 500:NEXT W
```

demnach einer halben Note,

```
FOR W=1 TO 250:NEXT W
```

einer viertel Note usw... Wenn man nun ein Musikstück abspielen lassen möchte, stellt sich die Frage, wie und wo man die Noten und ihre jeweilige Dauer ablegt. Am einfachsten geht das in DATA-Zeilen. Die Noten und ihre Dauer werden jeweils paarweise hintereinander abgelegt.

Wie so etwas konkret aussieht, zeigt das folgende Programm. Bevor Sie das Programm abtippen bzw. starten, vergessen Sie bitte nicht, dafür zu sorgen, daß die Routinensammlung im Speicher zur Verfügung steht! Da ich ja Ihren persönlichen Musikgeschmack nicht kenne, habe ich in dem Programm bewußt darauf verzichtet, Ihnen irgendeine Melodie vorspielen zu lassen. Das Programm spielt so, wie es abgedruckt ist, einfach die gesamte Tonleiter von der tiefsten bis zur höchsten Note. Natürlich können Sie die vorhandenen Noten aber sehr leicht durch die Noten einer Ihrer Lieblingsmelodien austauschen.

Das Programm soll Ihnen sozusagen als "Grundgerüst" für eigene Experimente dienen. Vielleicht probieren Sie auch einmal ein anderes Musikinstrument aus. Im Listing "voreingestellt" ist eine Flöte.

```
100 rem programm: musik
110 sys 40000:rem sid initialisieren
120 sys 40013,15:rem volle lautstaerke
125 :
130 rem musikinstrument: floete
140 sys 40238,1,1:rem wellenform dreieck
150 sys 40203,1,8,5,15,8:rem huellkurve
160 :
170 sys 40328,1:rem stimme ein
175 :
180 read nt$:rem note lesen
190 if nt$="ende" then 250:rem fertig
190 read dr:rem notendauer
200 sys 40068,1,nt$:rem note setzen
210 for w=1 to dr:next w:rem warteschleife
220 goto 180:naechste Note
240 :
250 sys 40338,1:rem stimme aus
260 end
```

```
980 :
990 :
1000 rem musiknoten und spieldauer
1010 data c0,1000,c#0,1000,d0,1000,d#0,1000,e0,1000,f0,1000
1020 data f#0,1000,g0,1000,g#0,1000,a0,1000,a#0,1000,h0,1000
1010 data c1,1000,c#1,1000,d1,1000,d#1,1000,e1,1000,f1,1000
1020 data f#1,1000,g1,1000,g#1,1000,a1,1000,a#1,1000,h1,1000
1010 data c2,1000,c#2,1000,d2,1000,d#2,1000,e2,1000,f2,1000
1020 data f#2,1000,g2,1000,g#2,1000,a2,1000,a#2,1000,h2,1000
1010 data c3,1000,c#3,1000,d3,1000,d#3,1000,e3,1000,f3,1000
1020 data f#3,1000,g3,1000,g#3,1000,a3,1000,a#3,1000,h3,1000
1010 data c4,1000,c#4,1000,d4,1000,d#4,1000,e4,1000,f4,1000
1020 data f#4,1000,g4,1000,g#4,1000,a4,1000,a#4,1000,h4,1000
1010 data c5,1000,c#5,1000,d5,1000,d#5,1000,e5,1000,f5,1000
1020 data f#5,1000,g5,1000,g#5,1000,a5,1000,a#5,1000,h5,1000
1010 data c6,1000,c#6,1000,d6,1000,d#6,1000,e6,1000,f6,1000
1020 data f#6,1000,g6,1000,g#6,1000,a6,1000,a#6,1000,h6,1000
1010 data c7,1000,c#7,1000,d7,1000,d#7,1000,e7,1000,f7,1000
1020 data f#7,1000,g7,1000,g#7,1000,a7,1000,a#7,1000
1030 data ende:rem endemarkierung
```

8. Die speziellen Schnittstellen

In diesem Kapitel möchte ich kurz auf die etwas spezielleren Anschlüsse des Commodore 64 eingehen. Wie man beispielsweise einen Drucker oder eine Floppy ansteuert, wissen Sie ja in der Zwischenzeit. Wie läßt sich aber zum Beispiel ein Joystick in eigenen Programmen abfragen?

8.1 Die Joystick-Anschlüsse

Nicht nur in Spielen, auch in Anwendungsprogrammen läßt sich ein Joystick durchaus sinnvoll einsetzen, was ja nicht zuletzt GEOS auf eindrucksvolle Weise demonstriert. Die Abfrage der beiden "Joystick-Ports" gestaltet sich sehr einfach. Sehen wir uns das ganze zunächst einmal in BASIC an. Als erstes muß die Tastaturabfrage gesperrt werden:

```
10 poke 56322,224:rem tastaturabfrage sperren
```

Anschließend wird das zu dem betreffenden Joystick-Port gehörende Register augelesen. Es enthält alle Informationen, die wir benötigen. Das Register für den Joystick-Port 1 hat die Adresse 56.321, das Register für Port 2 die Adresse 56.320:

```
20 jt=peek(56321):rem Joystick-Port1 abfragen
```

Jetzt können wir die Tastaturabfrage wieder freigeben:

```
30 poke 56322,255
```

und dann den aus dem Register ausgelesenen Wert analysieren. Für uns interessant sind die ersten fünf Bits:

```
Bit0: Bewegung nach oben
Bit1: Bewegung nach unten
Bit2: Bewegung nach links
Bit3: Bewegung nach rechts
Bit4: Feuerknopf
```

Falls das betreffende Bit gelöscht ist, hat die entsprechende Operation stattgefunden. Oft ist es zweckmäßig, die Richtungsänderungen nicht mit "oben", "unten" usw. zu bezeichnen, sondern die entsprechenden Himmelsrichtungen zu wählen:

```
Oben:   Nord
Unten:  Süd
Rechts: Ost
Links:  West
```

Die "Zwischenrichtungen" sind in vielen Fällen von Bedeutung:

```
Oben und rechts:  Nordost
Oben und links:   Nordwest
Unten und rechts: Südost
Unten und links:  Südwest
```

Eine komplette Abfrage sieht dann so aus:

```
100 rem nord
110 if (jt and 1)=0 then gosub 1000
120 rem nordost
130 if (jt and 9)=0 then gosub 2000
140 rem ost
150 if (jt and 8)=0 then gosub 3000
160 rem südost
170 if (jt and 10)=0 then gosub 4000
180 rem süd
190 if (jt and 2)=0 then gosub 5000
200 rem südwest
210 if (jt and 6)=0 then gosub 6000
220 rem west
230 if (jt and 4)=0 then gosub 7000
240 rem nordwest
250 if (jt and 5)=0 then gosub 8000
260 rem feuerknopf
270 if (jt and 16)=0 then gosub 9000
290 :
300 goto 10:rem naechster durchgang
```

Natürlich kann man die Feuerknopfabfrage auch mit einer der Richtungsabfragen verknüpfen. Wollen Sie zum Beispiel wissen, ob der Joystick bei gedrücktem Feuerknopf nach "südost" bewegt wurde, so müßte die Abfrage so aussehen:

```
280 if (jt and 26)=0 then gosub .....
```

Sie sehen, das ganze ist sehr flexibel. Eines sollten Sie aber beachten: Da zum Abfragen der Joystick-Ports die Tastaturabfrage gesperrt werden muß, besteht keine Möglichkeit, die Abfrageschleife durch die <Stop>-Taste zu unterbrechen! Starten Sie also das Programm und befindet sich in dem betreffenden Port gar kein Joystick, so kommen Sie aus der Programmschleife nicht mehr heraus. Dann hilft nur noch ein RESET. Auch in Assembler läßt sich die Joystick-Abfrage relativ leicht realisieren:

```
100 ;tastatur sperren
100 abfrage  lda #224
110          sta 56322
120 ;register von joystick-port2 auslesen
    ;und zwischenspeichern
130          lda 56320
140          sta 02
150 ;tastatur freigeben
160          lda #255
170          sta 56322
180 ;
190 ;register-wert analysieren
210          lda #1
220          and 02
230          beq nord
210          lda #9
220          and 02
230          beq nordost
210          lda #8
220          and 02
230          beq ost
210          lda #10
220          and 02
230          beq südost
210          lda #2
220          and 02
230          beq süd
210          lda #6
220          and 02
230          beq südwest
210          lda #4
220          and 02
230          beq west
210          lda #5
220          and 02
230          beq nordwest
210          lda #16
220          and 02
230          beq knopf
240          jmp abfrage
```

8.2 Der User-Port

Der "User-Port" ist eine frei programmierbare Schnittstelle, die - wie der Name schon sagt - zum Anschluß anwenderspezifischer Peripheriegeräte dient. Der User-Port läßt sich im wesentlichen auf dreierlei Weise nutzen:

Als programmierbare Steuerschnittstelle für Hardware-Schaltungen

Von der einfachen Diodenschaltung über eine Rolladensteuerung bis hin zur Robotersteuerung (dafür gibt es von Fischertechnik sogar spezielle Bausätze) - mit dem User-Port läßt sich alles messen, steuern und regeln.

Dazu sind natürlich gute Elektrotechnik-Kenntnisse erforderlich. Außerdem benötigt man detaillierte Informationen über den Hardware-Aufbau des Commodore 64. Es würde an dieser Stelle zu weit führen, auf weitere Programmierdetails einzugehen.

Als Centronics-Schnittstelle

Ein Drucker mit einer sogenannten "Centronics-Schnittstelle" läßt sich ebenfalls am User-Port anschließen. Dazu benötigt man nur ein geeignetes Verbindungskabel (in jedem guten Fachgeschäft erhältlich) sowie spezielle Steuerungs-Software, die die Centronics-Schnittstelle am User-Port realisiert.

Auch bei der Steuerungs-Software kann man sich die Eigenprogrammierung, die gute Hardware-Kenntnisse erfordert, ersparen. Jede gute Textverarbeitung ist heutzutage in der Lage, eine Centronics-Schnittstelle am User-Port anzusteuern. Außerdem verfügen auch die meisten Steckmodule über eine entsprechende Implementierung. Diese Modul-Centronics-Schnittstelle steht in der Regel von allen Programmen aus zur Verfügung, so daß man damit beispielsweise auch seine BASIC-Programme über den User-Port ausdrucken lassen kann.

Als RS-232-Schnittstelle

Die vielleicht interessanteste Schnittstelle, die man am User-Port realisieren kann, ist eine sogenannte RS-232-Schnittstelle. Dabei handelt es sich um eine Schnittstelle zur seriellen Datenübertragung. Die RS-232-Schnittstelle wird zum Beispiel bei der Datenfernübertragung verwendet. Dabei werden dann zwei Rechner über ihre RS-232-Schnittstellen und das öffentliche Telefonnetz miteinander verbunden, um Daten auszutauschen. Genauso gut kann man zwei Rechner aber auch direkt miteinander verbinden, ohne extra das Telefonnetz zu bemühen.

So könnten Sie beispielsweise zwei Commodore 64 miteinander koppeln oder auch Daten zwischen Ihrem Commodore 64 und einem PC, einem Atari St oder einem Amiga austauschen. Zunächst benötigen Sie natürlich ein passendes Verbindungskabel. Im einschlägigen Fachhandel gibt es entsprechende Kabel für fast alle Zwecke. Man ist also nicht unbedingt gezwungen, selbst zum Lötkolben zu greifen.

Außerdem gibt es von Commodore auch ein spezielles RS-232-Steckmodul, das auf den User-Port gesteckt werden kann. Hat man dieses Modul zur Verfügung, dann genügt ein einfacheres Kabel. Die RS-232-Schnittstelle ist wie gesagt eine serielle Schnittstelle, d.h., die Daten werden Bit für Bit hintereinander übertragen. Eine Centronics-Schnittstelle beispielsweise arbeitet dagegen parallel. In einem Durchgang werden jeweils acht Bits übertragen. Diese Form der Datenübertragung ist natürlich wesentlich schneller als die serielle Übertragung.

Der große Nachteil dabei ist aber die mangelnde Datensicherheit bei größeren Entfernungen. Ab einer Kabellänge von etwa zwei Metern beeinflussen sich die einzelnen Leitungen gegenseitig, so daß es zu Störungen bei der Übertragung kommen kann. Speziell abgeschirmte Kabel schaffen dem zwar Abhilfe, für größere Entfernungen sind derartige Kabel aber viel zu teuer. Möchte man die Daten über das Telefon übertragen, so scheidet eine parallele Übertragung von vornherein aus. Das Telefonnetz arbeitet nämlich mit nur jeweils zwei Leitungen zwischen zwei Anschlüssen, das sind sechs Leitungen zuwenig.

Die Programmierung der RS-232-Schnittstelle gestaltet sich fast so einfach wie beispielsweise die Ansteuerung der Floppy oder des Druckers. Man muß lediglich einige Parameter mehr einstellen. Das Betriebssystem des Commodore 64 enthält nämlich bereits sämtliche Routinen zur Ansteuerung der RS-232-Schnittstelle. Die Arbeit gliedert sich in drei Phasen:

- Öffnen der Schnittstelle mittels einer OPEN-Anweisung.
- Lesen und/oder Schreiben von Daten mit INPUT#, GET# bzw. PRINT#.
- Schließen der Schnittstelle mittels einer CLOSE-Anweisung.

Um den Aufbau der OPEN-Anweisung zu verstehen, müssen wir uns noch ein wenig mit dem Übertragungskonzept befassen. Zur Erhöhung der Übertragungssicherheit werden jeweils acht zu übertragende Bits in ein Start-Bit (vor dem ersten Daten-Bit) und in ein oder zwei Stop-Bits (hinter dem letzten Daten-Bit) "eingebettet".

Wahlweise läßt sich auch noch ein sogenanntes Paritäts-Bit einfügen. Dieses Bit gibt an, ob die Anzahl der Einsen in den Daten-Bits gerade oder ungerade ist. Veranschaulichen wir uns das ganze einmal grafisch:

0	1	2	3	4	5	6	7	0	1	2	....
Datenbyte 1								Datenbyte 2			

Die Nummern 0 bis 7 stehen jeweils für die einzelnen Daten-Bits eines Bytes. Vor jedes Daten-Byte wird nun ein Start-Bit "S" und hinter jedes Daten-Byte ein oder zwei Stop-Bits "S1" und "S2" eingefügt. Für ein einzelnes Byte sieht das dann so aus:

S	0	1	2	3	4	5	6	7	S1

bzw.:

S	0	1	2	3	4	5	6	7	S1	S2

Das Paritäts-Bit wird anstelle des 7. Daten-Bits eingefügt:

S	0	1	2	3	4	5	6	P	S1

bzw.:

S	0	1	2	3	4	5	6	P	S1	S2

Ein weiterer wichtiger Punkt bei der Datenübertragung ist die Übertragungsgeschwindigkeit. Während man bei der Floppy oder beim Drucker keine Möglichkeit hat, die Übertragungsgeschwindigkeit zu variieren (von Floppy-Speedern einmal abgesehen), läßt sie sich bei der RS-232-Schnittstelle auf Bits genau einstellen. Die Einheit, in der die Übertragungsgeschwindigkeit gemessen wird, heißt "Baud" und steht für "Bits je Sekunde". Eine Übertragungsgeschwindigkeit von 2.400 Baud bedeutet also, daß je Sekunde 2400 Bits übertragen werden.

Schließlich muß noch angegeben werden, in welcher Übertragungsart gearbeitet werden soll, "vollduplex" oder "halbduplex". Bei Vollduplex-Betrieb werden die Daten in beiden Richtungen mit derselben Geschwindigkeit gesendet, bei Halbduplex-Betrieb, erfolgt die Datenübertragung in eine Richtung wesentlich verlangsamt. In der Regel arbeitet man im Vollduplex-Betrieb.

Der letzte anzugebende Parameter betrifft den sogenannten "Handshake-Modus". Zur Auswahl stehen "3-Draht-Handshake" und "X-Draht-Handshake". Beim 3-Draht-Handshake, dem einfacheren der beiden Modi, erfolgt bei der Datenübertragung keine Kontrolle, ob die gesendeten Daten auch tatsächlich beim Empfänger angekommen sind. Außerdem besteht für den empfangenden Rechner keine Möglichkeit, die Datenübertragung

anzuhalten. Diese beiden Nachteile hat der X-Draht-Handshake nicht.

Die Art des Handshakes hängt von der Verdrahtung des Verbindungskabels ab. Bei einfachen Rechner-Rechner-Verbindungen reicht der 3-Draht-Handshake in der Regel aus. Bei einer Verbindung Rechner-Modem beispielsweise benötigt man dagegen den X-Draht-Handshake.

Damit haben wir alle Parameter beieinander. Doch wie wählt man diese nun aus und wie teilt man sie dem Commodore 64 mit? Dazu verfügt die RS-232-Schnittstelle über zwei Register, ein "Kontroll-Register" und ein "Befehls-Register".

Die Bits 0 bis 3 des Kontroll-Registers sind für die Übertragungsgeschwindigkeit, die "Baudrate", zuständig:

Baudrate	3	2	1	0	Wert
50	0	0	0	1	1
75	0	0	1	0	2
110	0	0	1	1	3
134.5	0	1	0	0	4
150	0	1	0	1	5
300	0	1	1	0	6
600	0	1	1	1	7
1200	1	0	0	0	8
1800	1	0	0	1	9
2400	1	0	1	0	10

Die Bits 5 und 6 des Kontroll-Registers sind für die Anzahl der Daten-Bits zuständig:

Daten-Bits	6	5	Wert
8	0	0	0
7	0	1	32
6	1	0	64
5	1	1	96

Bit 7 des Kontroll-Registers bestimmt die Anzahl der Stop-Bits:

Stop-Bits	7	Wert
1	0	0
2	1	128

Das Bit 0 des Befehls-Registers bestimmt die Handshake-Art:

Handshake	0	Wert
3-Draht	0	0
X-Draht	1	1

Das Bit 4 des Befehls-Registers bestimmt die Übertragungsart:

Art	4	Wert
Vollduplex	0	0
Halbduplex	1	16

Bit 5 bis 7 sind für die Art der Paritätsprüfung zuständig:

Prüfung	7 6 5	Wert
keine	0 0 0	0
ungerade	0 0 1	32
gerade	1 0 1	96
keine	0 1 0 8. Daten-Bit = 1	160
keine	1 1 1 8. Daten-Bit = 0	224

Beide Register-Werte werden in der OPEN-Anweisung als "File-Namen" übergeben und zwar in der Reihenfolge Kontroll-Register-Befehls-Register. Nehmen wir einmal an, Sie möchten die RS-232-Schnittstelle mit den folgenden Parametern öffnen:

```
300 Baud
7 Bit ASCII Daten
1 Stop-Bit
keine Parität
8. Daten-Bit immer 1
Vollduplex
3-Draht-Handshake
```

Das sieht dann so aus:

```
100 lf=2:rem logische file-nummer (frei wählbar)
110 ga=2:rem geraeteadresse der rs-232-schnittstelle
120 sa=0:sekundaeradresse (ohne bedeutung)
130 :
140 bd=6:rem baudrate (1-10)
150 db=0:rem daten-bit-anzahl (0,32,64,96)
160 sp=0:rem stop-bit-anzahl (0,128)
170 :
180 hd=0:rem handshake (0,1)
190 ua=0:rem uebertragungsart (0,16)
200 pr=160:paritaet (0,32,96,160,224)
210 :
220 open lf,ga,sa,chr$(bd+db+sp)+chr$(hd+ua+pr)
```

Dann kann man mit INPUT#2 und GET#2 Daten lesen oder mit PRINT#2 schreiben. CLOSE 2 beendet die Übertragung.

8.3 Der Expansion-Port

Der "Expansion-Port" dient ausschließlich dazu, sogenannte Steckmodule aufzunehmen. Sobald der Commodore 64 eingeschaltet wird, überprüft die dabei ablaufende RESET-Routine, ob ein Modul eingesteckt ist. Dazu muß im Modul-Speicher die Kennung "CBM80" abgelegt sein. Ist das der Fall, so wird die Rechnerkontrolle an die Modul-Software übergeben. Diese Modul-Software liegt in der Regel in sogenannten EPROM vor. EPROM bedeutet "Erasable Programmable Read Only Memory", also lösch- und beschreibbarer ROM-Speicher. Um die Software in das EPROM zu bekommen, ist ein spezielles EPROM-Programmiergerät erforderlich, mit dessen Hilfe die Software in das EPROM "gebrannt" wird. EPROM-Programmiergeräte zum Betrieb am Commodore 64 und die zugehörigen EPROM gibt es bei verschiedenen Spezialanbietern zu kaufen. Für die hobbymäßige Anwendung dürfte das ganze allerdings zu teuer sein.

9. Pflegen und Reparieren - Floppy 1541 und C64

Die Floppy 1541 dient Ihnen zur Sicherung von Daten, wobei die Vorgänge schneller ablaufen als z.B. bei der Datasette. Sie erhalten im folgenden Hinweise zur Pflege und Reparatur des Gerätes.

9.1 Einleitung

Vor dem Aufdrehen der Gehäusebefestigungsschrauben sollten Sie die 220-Volt-Versorgungsleitung und das Seriellbuskabel von der Floppy abziehen. Für das Öffnen der vier Gehäuseschrauben sollten Sie einen Kreuzschlitz-Schraubendreher der Größe Nr. 2 verwenden, welchen Sie auch noch weiterhin benötigen werden.

Vorsicht! Die Digitalplatine Ihres 1541 ist mit Bausteinen bestückt. Die Bausteine sind sehr empfindlich gegen statische Aufladungen die durch den Menschen übertragen werden können. Gehen Sie deshalb mit ihrer Elektronikplatine sehr behutsam um. Ein Kurzschluß, der durch ihre Hände oder herumliegendes Werkzeug verursacht wird, kann ebenfalls zum "Sterben" eines ICs führen.

9.1.1 Prüfen ohne Werkzeug

Die hier beschriebenen Vorgehensweisen sind für alle Laufwerkstypen, natürlich mit Ausnahme der 3,5-Zoll-Floppy 1581, gültig. Die im Kapitel "Überwachungsdienst alle 4 Monate" (Wärmeentwicklung und Geräuschentwicklung des Transformators) beschriebenen Vorgehensweisen beziehen sich nur auf Laufwerke, in denen das Netzteil fest eingebaut ist.

Beim Kauf Ihres Diskettenlaufwerks haben Sie zum Gerät zwei Disketten erhalten. Die eine Diskette enthält u.a. auch das Programm PERFORMANCE TEST und DISPLAY T&S sowie weitere nützliche Programme. Das Programm PERFORMANCE TEST fordert Sie auf, eine neue Diskette einzulegen und den Test mit <Return> zu starten. Daraufhin wird die Diskette formatiert und anschließend ein Schreib-/Lesetest durchgeführt. Dieser druckt nach Beendigung PERFORMANCE TEST OK oder eine Fehlermeldung mit Angabe der Spur und dem Sektor, in dem der Fehler auftrat, auf den Bildschirm. Eine Liste der Bedeutungen der Fehlermeldungen finden Sie im Anhang.

Ein READ ERROR in der Spur 35 weist auf eine Dejustage der Spur-1-Einstellung hin, genauso ein READ ERROR in der Spur 1 oder in mehreren Spuren. Die Besonderheit beim PERFORMANCE TEST ist, daß die am äußeren Rand der Disk liegende Spur als Spur 35 bezeichnet wird. Sie ist aber mit der Spur 1 identisch. Im Klartext: Die Spur 1 befindet sich außen auf der Diskette und ist im PERFORMANCE TEST als Spur 35 gekennzeichnet.

Das Programm "DISPLAY T&S" sollten Sie vor Beginn ihrer Tätigkeiten an ihrer Floppy 1541 testen. Sollten Sie dieses Programm nicht besitzen, rate ich Ihnen, sich dieses Programmm bei Ihrem Fachhändler zu besorgen. Sollte Ihre 1541 überhaupt nicht mehr laden, sollten Sie sich von einem Bekannten einen Kassettenrekorder 1530 leihen, und sich das Programm "DISPLAY T&S" auf Kassette überspielen. Noch besser wäre es jedoch, es leiht Ihnen jemand eine 1541.

9.1.2 Werkzeuge und andere Hilfen

Werkzeuge für die Mechanik

- Kreuzschlitz-Schraubendreher Größe 2
- Schraubendreher normal/klein Breite 2.5mm und normal/groß Breite ca. 8 mm
- Spitzzange klein
- Elektroniker-Mikroschere

Hilfsmittel

- Sekundenkleber, Panzerkleber oder Nagellack
- Zwei Disketten des 1541 Diskettenlaufwerks, die in einem Laufwerk formatiert wurden, das sich in einem guten Zustand befand (für R/W-Test und Spur-1-Justage), oder bereits ältere Disketten vor der Dejustierung ihres Laufwerks.
- Nähmaschinenöl und säurefreies Fett (Vaseline)
- Spiritus oder ähnliches (fettfrei)
- Wattestäbchen

Werkzeuge für die Elektronik

- Vielfachmeßgerät (für Spannungs-/Widerstandsmessungen)

9.1.3 Garantie und Reparaturen

Natürlich können einmal Schäden oder Störungen Sie zwingen mit Ihrem 1541 Laufwerk eine Werkstatt aufzusuchen. Dann könnte ein Spezialist in manchen Fällen gezielter helfen. Gehen Sie dann zu einem Vertragshändler der Fa. Commodore, da dieser in der Regel die erforderlichen Ersatzteile auf Lager hat. Solche Service-Werkstätten sind auf ihrem Spezialgebiet mit den Geräten vertraut.

Aber vielleicht führt es wegen Arbeitsüberlastung der Werkstatt zu Verzögerungen, dann können Sie die erforderlichen Reparaturen oder Wartungsarbeiten eventuell selbst durchführen und die nötigen Ersatzteile dort kaufen.

Garantie-Ansprüche

Sie haben auf Ihre 1541-Diskettenstation eine Garantie, die Ihnen Ihr Händler oder Kaufhaus, bei dem Sie Ihr Gerät erworben haben, gewährt. Dieser Garantie-Anspruch verfällt, wenn Sie in diesem Zeitraum in Ihre 1541 eingreifen. Dies gilt auch bei anderen Geräten, wie beim 64'er oder Druckern. Schon ein Ändern der Geräteadresse, was Sie selbst durchführen können, läßt den Garantie-Anspruch erlöschen.

Selbst ist der Mann

Oben aufgeführte Hinweise sollen Sie jedoch nach der Garantiezeit nicht daran hindern, Zeit und Geld zu sparen. Prüfen Sie vor einer Reparatur Ihre eigenen Kenntnisse und die Ihnen zur Verfügung stehenden Mittel, bevor Sie ans Werk gehen. Bei allen Arbeiten, die Sie durchführen, ist Sorgfalt und Feingefühl die oberste Pflicht.

Sollten Sie das Gefühl haben, daß es für Sie zu schwierig wird, können Sie immer noch eine Fachwerkstatt aufsuchen.

9.2 Pflegen des Laufwerks

Das Wohlergehen Ihres 1541 richtet sich - wie auch bei anderen technischen Geräten - nach der Pflege, die Sie ihm zuteil werden lassen. Regelmäßige Überprüfungen, die sich aus Funktionskontrollen und Pflegearbeiten zusammensetzen, sind ebenso wichtig, wie etwa der Kundendienst bei einem Auto.

9.2.1 Wartung - wann und wo?

Da vom Hersteller keine Wartungsintervalle festgelegt sind, soll die unten aufgeführte Tabelle Ihnen bei der Pflege Ihres 1541 Diskettenlaufwerks behilflich sein.

Bei näherer Betrachtung stellt man fest, daß viele Dinge in wenigen Handgriffen zu erledigen sind und einzelne Punkte bei etwas routinierter Beziehung zu Ihrem Diskettenlaufwerk sogar ohne Anleitung durchgeführt werden können. Dennoch sollten Sie zur erstmaligen Pflege auch die Kurzbeschreibungen der einzelnen Wartungsarbeiten durchlesen.

Pflegearbeiten alle 4 Monate

- Reinigen des Schreib-Lesekopfes
- Laufwerk entstauben
- Schmieren der Führungswellen

Überwachungsdienst alle 4 Monate:

- Wärmeentwicklung im Gerät überprüfen
- Geräuschentwicklung des Transformators
- Stroboskopscheibe überprüfen

Pflegearbeiten alle 12 Monate:

- Antriebsriemen im Laufwerk ersetzen
- Andrucksfilz austauschen
- Schmieren der Laufwerksmechanik
- Geschwindigkeit überprüfen

Pflegearbeiten alle 18 Monate:

- Schreib-/Lesekopfeinstellung korrigieren
- Spur-1-Einstellung prüfen

Es soll Anwender geben, die ihr Gerät sorglos drauflos benutzen, ihre Diskettenstation nicht pflegen, nicht reinigen, geschweige denn an eine Wartung oder Nachjustierung denken. Aus Furcht vor Werkstattkosten oder Ausfall des Gerätes führen andere hingegen zuviel des guten durch, so daß zum Schluß das Diskettenlaufwerk, von Öl verklebt, nicht arbeiten kann und die Köpfe der Schrauben zur Befestigung der Gehäuseteile und der Elektronikplatine abgenutzt oder abgedreht sind. Der Mittelweg ist hier meist am besten. Die oben genannten Intervalle wurden aus der täglichen Praxis mit 1541-Diskettenlaufwerken abgeleitet.

9.2.2 Vorbereitung zur Wartung

Bevor man das Gerät zum Öffnen der Gehäuseschrauben umdreht, sind die beiden Stecker auf der Geräterückseite, 220-Volt-Anschluß und serieller IEEE-488-Bus, abzuziehen. Drehen Sie nun das Gerät um, so daß es auf der Geräteoberseite liegt. Sie benötigen zuerst nur einen Kreuzschlitz-Schraubendreher, um die Gehäusebefestigungsschrauben zu lösen. Damit Sie keine Schrauben verlieren oder verwechseln, sollten die Schrauben in mehreren kleinen Schachteln zusammengehörend aufbewahrt werden. So können Sie vermeiden, daß die verschiedenen

Schrauben vertauscht werden und daß Sie nicht versehentlich nach Beendigung aller Arbeiten Schrauben an einer falschen Stelle verwenden.

Anschließend wird das Gerät wieder umgedreht, das Gehäuseoberteil abgehoben und zur Seite gelegt. Je nach Laufwerkstyp sieht man die Befestigungschrauben, die das Blechchassis mit dem Gehäuseunterteil verbinden. Diese Schrauben werden dann mit dem Schraubendreher entfernt und separat aufbewahrt.

Die Steckverbindung des Kabels, das zur Leuchtdiode in der Gehäusefront führt, ist von der dreipoligen Stiftleiste auf der Digitalplatine abzuziehen. Steckverbindungen, welche getrennt werden müssen, sollte man mit Farbstift, kleinen Aufklebern oder Tipp-Ex markieren, wobei die Aufkleber die eleganteste Methode darstellen, da diese z.B mit Zahlen beschriftet werden können. Durch Kennzeichnung der Steckverbindungen vermeiden Sie, daß Stecker beim Zusammenbauen nach beendeter Arbeit falsch oder vertauscht aufgesteckt werden. Ein Verpolen der Anschlüsse kann dazu führen, daß IC oder eine ganze Baugruppe beschädigt werden.

Für Justagearbeiten am Schreib-/Lesekopf oder Korrekturen der Geschwindigkeit des Diskettenantriebsmotors, sind keine weiteren Schrauben zu entfernen oder Steckverbindungen zu lösen. Das Chassis wird hochkantig auf die linke Seitenfläche gestellt, so daß der Diskettenschacht vor Ihnen liegt und die Geräteunterseite von rechts zugänglich ist.

Damit das Gerät senkrecht steht, sollte man einen ca. 15 mm starken Gegenstand an den gegenüberliegenden Blechlaschen an beiden Geräteenden unterlegen. Die nächsten Schritte sind nur auszuführen, wenn direkt an der Unterseite des Laufwerks gearbeitet wird.

Um die Pflegearbeiten und Justagen durchführen zu können, müssen die Befestigungsschrauben der Digitalplatine entfernt und vorerst alle Steckverbindungen der Platine getrennt werden.

Die Digitalplatine legen Sie vorerst an einer gefahrenlosen Stelle ab. Vorsicht: Elektrostatische Aufladungen schädigen die Digitalplatine. Jetzt sieht man das eigentliche Laufwerk, das mit Schrauben befestigt ist. Nun kann man die Schrauben auf der linken und rechten Seite jeweils mit einen Kreuzschraubendreher entfernen. Anschließend heben Sie das Laufwerk heraus und stellen es vorsichtig auf Ihrem Arbeitstisch ab. An dieser Stelle soll nicht vergessen werden, Sie darauf hinzuweisen, daß verschiedene Teile der Laufwerksmechanik sehr empfindlich sind.

Wenn man eine Justage des Spur-1-Stoppanschlags oder im Anschluß dazu eine Schreib-/Lesekopf-Justage ausführen will, muß die Digitalplatine angeschlossen werden. Um den Spur-1-Stoppanschlag einstellen zu können, sind die Geräteteile so anzuordnen, daß man ein funktionsfähiges Diskettenlaufwerk hat, wobei man aber an die Befestigungsschraube des Kopfanschlages für die Spur 1 herankommen muß. Dazu muß man keine Sondervorrichtung bauen, sondern nur das Blechchassis in das Gehäuseunterteil (ohne Laufwerk) einsetzen, wobei es nicht angeschraubt wird. Schraubt man die Digitalplatine nun auf das Chassis und verbindet den Netzteilstecker vom Transformator, hat man bereits das funktionsfähige Gerät ohne Laufwerk. Stellen Sie nun das Gerät ohne Laufwerk mit dem Diskettenschacht nach vorne, und legen Sie eine Diskettenbox für 10 Disketten oder ein gleich dickes Buch links neben das Gerät.

Ungefähr in der Mitte des Gerätes stellen Sie das Laufwerk senkrecht auf die links liegende Unterlage mit den Steckern des Laufwerks zur Digitalplatine hin. Positionieren Sie das Laufwerk so, daß es mit Steckern des Laufwerks so nahe wie möglich an den dazugehörigen Stiftleisten auf der linken Seite der Digitalplatine steht. Das Laufwerk liegt jetzt im rechten Winkel zum Gehäuse, so daß die Rückseite des Laufwerks, aus dem die Kabel mit den Verbindungen herausgeführt sind, an den Stiftleisten liegen.

Die Stecker des Laufwerks kann man jetzt mit den dazugehörigen Stiftleisten der Digitalplatine verbinden. Eine Erleichterung erfahren Sie, wenn die zusammengehörenden Steckverbindungen bereits markiert sind. Zuletzt wird der schwarze Stecker des

Schreib-/Lesekopfes mit der Platine verbunden, da das Kabel den weitesten Weg zur Digitalplatine hat (Vorsicht vor Verpolung).

Nun haben das Laufwerk und der elektronische Teil des 1541 eine Position erreicht, in der man den Spur-1-Anschlag einstellen kann. Schreib-/Lesekopfjustage nach einer Spur-1-Einstellung. Dreht man nun das Laufwerk um 90 Grad, so daß es mit der Seitenfläche auf dem Buch oder der Diskettenbox liegt, kann man in dieser Position die Justage der R/W-Kopfes durchführen, wie es bereits weiter vorne beschrieben wurde.

In dieser Position steht das Laufwerk fest auf der Unterlage und ist von unten sehr gut zugänglich.

9.2.3 Der kleine Wartungsdienst

Sie sollten es unterlassen, eine Modifikation des Stopprings durchzuführen, indem Sie die Welle des Steppermotors (für Kopftransport) und den Stopring durchbohren und durch einen Splint sichern, wie es einmal in einer Zeitschrift zu lesen war.

Diese Warnung soll Sie aber von ihrem Vorhaben, ihr Diskettenlaufwerk 1541 selbst zu warten, nicht abhalten.

Pflegearbeiten alle 4 Monate - selbst ist der Mann

Reinigen des R/W-Kopfes

Heben Sie die Andruckplatte am R/W-Kopf an, und wischen Sie mit einem in Spiritus getränkten Wattestäbchen über die rechteckige Keramikplatte des Schreib-/Lesekopfes.

Laufwerk entstauben

Verwenden sie einen Pinsel für die Entstaubung des Laufwerks.

Schmieren der Führungswelle

Über die beiden Führungswellen gleitet der Kunststoffträger des R/W-Kopfes, der auf 3 Punkten gelagert ist. Für die Schmie-

rung dieser Welle sollten Sie säurefreies Fett (z.B Vaseline) verwenden, da es den Kunststoff nicht angreift. Benutzen Sie ein Wattestäbchen mit Vaseline, und streichen Sie damit über die beiden Wellen. Bereits ein dünner Fettfilm sorgt für ausreichende Schmierung.

Überwachungsdienst

Wärmeentwicklung im Gerät kontrollieren

Ihr 1541 benötigt eine Gleichspannung von 5 Volt für die Digitalplatine und 12 Volt "DC" für das Laufwerk.

Diese Spannungen müssen jedoch aus unserer Netzspannung von 220 Volt Wechselstrom gewonnen werden. Dafür enthält unser 1541 den Transformator T1, der durch sein Übersetzungsverhältnis diese 220 Volt/AC in eine kleinere Wechselspannung transformiert. Der Transformer stellt zwei der Wechselspannungen zur Verfügung, die getrennt über den Stecker den Brückengleichrichtern zugeführt werden (DC = Gleichspannung, AC = Wechselspannung). Jeder Gleichrichter für sich erzeugt eine Gleichspannung, die über zwei Kondensatoren zur Glättung der Spannung an die Spannungsregler für 12 Volt/DC oder für 5 Volt/DC geführt wird. Diese Spannungsregler strahlen je nach Leistungsaufnahme Ihres Gerätes mehr oder weniger Wärme ab.

Bei direktem Betrieb, wenn sich der Diskettenantriebsmotor und R/W-Kopftransportmotor dreht, wird eine höhere Leistung aufgenommen als im indirekten Betrieb. Über zwei Kühlkörper und eine Verbindung zum Blechchassis des Gerätes wird die erzeugte Wärme abgeführt. Ist einer der Spannungsregler defekt oder wird aus irgendeinem Grund zuviel Leistung aufgenommen, so kann es dazu führen, daß die Kühlung nicht ausreicht. Deshalb sollten Sie regelmäßig die Wärmeentwicklung überprüfen, indem Sie mit einem Finger die Gehäusoberflächen des Spannungsreglers berühren. Temperaturen von 30 bis ca. 55 Grad sind als normal anzusehen. Ebenso ist bei den Gleichrichtern zu verfahren. Sollten sie so heiß sein, daß Sie sich die Finger verbrennen oder daß an der Platine Verfärbungen erkennbar sind, liegt ein Defekt vor.

Eine weitere Wärmequelle kann der Transformer sein, wenn er im Laufwerksgehäuse fest eingebaut ist und die Isolierung des verwendeten Kupferdrahtes schadhaft geworden ist. Dadurch kann es zu Kurzschlußströmen im Transformer kommen.

Diese Kurzschlußströme können den Transformer so weit erhitzen, bis dieser eine Temperatur von über 40 Grad Celsius erreicht und das Gerät somit aufheizt. In diesem Fall ist der Transformer zu ersetzen, was jedoch sehr selten ist.

Geräuschentwicklung des Transformers

Wenn die Wicklungen des Kupferdrahtes im Transformer nicht richtig vergossen sind, kann es dazu führen, daß die Drähte in Resonanz geraten und Geräusche entwickelt werden. Dann entstehen Töne, die vom tiefen Brummen bis zu einem hohen Pfeifen reichen können. Erzeugt der Transformer Ihres Gerätes Geräusche oder neigt er zur Wärmeentwicklung, sollten Sie dies bei Ihrem Händler auch nach der Garantiezeit reklamieren. Ihr Fachhändler könnte in diesem Fall den Transformer beim Gerätehersteller als von Anfang an mangelhaftes Teil reklamieren. Eventuell bekommen Sie ihn ausgetauscht.

Überwachungsdienst: Stroboskopscheibe überprüfen

Damit sich das Antriebsrad bei Geräten ohne Sicherungsschraube nicht lösen kann, ist es zu verkleben. Dazu gibt man 1 - 2 Tropfen Sekunden- oder Panzerkleber auf die Verbindung der Welle und des Antriebsrades.

Hat sich die Antriebsscheibe von der Welle gelöst, ist von oben auf den Diskettenteller zu drücken und die Scheibe von der Unterseite des Laufwerks wieder auf die Welle zu pressen. Dazu sollte man keinen großen Hammer verwenden, sondern mit dem Griff des Schraubendrehers 2 - 3 mal leicht auf das Antriebsrad klopfen (auf keinen Fall zu fest).

Anschließend ist die Verbindung, wie oben beschrieben, mit Kleber zu sichern. Damit man den Diskettenteller niederdrücken kann, muß die Digitalplatine demontiert werden. Von einem

nachträglichen Einbau der Befestigungsschraube zur Sicherung der Antriebsscheibe ist abzuraten, da eine Beschädigung des Laufwerks bei der Durchführung der nötigen Arbeiten sehr wahrscheinlich ist.

9.2.4 Der große Wartungsdienst

Pflegearbeiten alle 12 Monate

Der Antriebsriemen ist aus Gummi hergestellt und unterliegt somit der Alterung gummihaltiger Produkte. Wie bei Autoreifen wird auch dieser Gummi brüchig, spröde und bekommt Risse. Ein gealteter Riemen kann zu einem hohen Verlust bei der Antriebsübertragung führen, was zur Folge hat, daß die Drehzahl der Diskette zu niedrig wird oder ständig ansteigt und abfällt.

Dieses Verhalten führt häufig zu READ ERROR? in den äußeren Spuren der Disketten. Ebenso macht sich dieser Effekt auch bei größeren Schreibzugriffen auf die Diskette bemerkbar. Eine typische Fehlermeldung ist dann die Fehlermeldung 21:

```
SYNCRONISATIONS Markierung auf der Diskette wird nicht gefunden.
```

Je ein Sektor auf der Diskette enthält zwei Sync-Markierungen, deren Bits alle "1" sind, also 1111111111 binär dargestellt. Die Spuren 1 bis 17 auf der Diskette enthalten 42 Sync-Zeichen pro Spur, wobei die Spuren 31 bis 35 nur noch 34 Sync-Zeichen je Spur enthalten.

Durch die höhere Anzahl von Sync-Zeichen in den äußeren Spuren erklären sich somit die Syncronisations-Probleme bei zu niedriger Drehzahl.

Antriebsriemen des Laufwerks ersetzen

Ziehen Sie den Antriebsriemen von den Riemenscheiben, und legen Sie anschließend den neuen Riemen so ein, daß die glänzende Seite auf den Antriebsscheiben zum Liegen kommt.

Legen Sie den Antriebsriemen zuerst um die kleinere der beiden Antriebsscheiben und ziehen danach den Riemen unter Drehen der größeren Scheibe auf. Die Demontage des Gehäuses ist wie in Unterkapitel 9.2.2 durchzuführen.

Andruckfilz austauschen

Bei Abnutzung des Andruckfilzes wird die Diskette nicht ausreichend gegen den Schreib-/Lesekopf gedrückt, was dazu führt, daß der Lesestrom im R/W-Kopf zu gering ist.

Aus diesen Informationen vom R/W-Kopf kann der Leseverstärker der Elektronikplatine kein ausreichendes Signal gewinnen, was ebenso zu READ und WRITE ERRORS führen kann.

Bei Abnutzung oder Vibration ("der Drive singt") muß der Andruckfilz bereits früher ausgetauscht werden. Mit einer Spitzzange wird die Halteklammer des Andruckfilzes zusammengedrückt und die Halteklammer herausgezogen. Anschließend wird der Andruckfilz aus der Halteklammer entfernt und der neue Filz in die Halteklammer gedrückt. Die Halteklammer wird daraufhin wieder in den Andruckhebel eingerastet.

Die beiden Ersatzteile erhalten Sie bei Ihrem Commodore-Fachhändler. Wenn Sie diese Teile selbst einbauen, sparen Sie bereits einige DM an Montagekosten die Ihnen in einer Fachwerkstatt angefallen wären.

Funktion des Andruckfilzes

Der Andruckhebel, der den Filz trägt, drückt die Diskette, die sich im Laufwerk befindet, mit einer Kraft von ca. 5 Gramm gegen den Schreib-/Lesekopf. Die Andruckfeder, die diese Kraft erzeugt ist in der Regel keine Fehlerursache.

Deshalb sollten an dieser Feder keine Änderungen vorgenommen werden. Die jeweilige Diskettenoberseite unterliegt sonst einer höheren Reibung durch den Andruckfilz. Sollten Sie Ihre Disketten beidseitig verwenden (wovon Sie niemand abbringen

will), wird auch die ansonsten unten liegende Seite der Reibung des Andruckfilzes ausgesetzt. Dieser Andruckfilz hat auf die oben liegende Diskettenseite die Wirkung einer Polierscheibe. Bei beidseitig verwendeten Disketten verringert sich dadurch die Haltezeit der Diskette.

Sie können das Abtragen der Magnetfolie am Andruckfilz Ihres Laufwerks durch eine bräunliche Färbung des Filzes erkennen. Bei Doppelkopf-Laufwerken kann diese Abnutzung der Diskettenoberseite nicht auftreten, da sich statt des Andruckfilzes ein zweiter Schreib-/ Lesekopf auf der Diskettenoberseite befindet.

Der Schreib-/Lesekopf hat durch seine keramische Beschichtung eine geringere Rauhtiefe, wodurch er eine wesentlich geringere Reibung erzeugt.

Das Speichermedium

Die Lebensdauer einer Diskette liegt bei ungefähr 10 Mio. Umdrehungen pro Spur beim Schreiben oder Lesen. Die Dicke der Magnetfolien beträgt ca. 80 Mikrometer. Die Magnetfolien sind mit einer speziellen Beschichtung versehen, die eine Oberflächenglätte sowie hervorragende Gleiteigenschaften erzeugt.

Die Lagertemperatur für Disketten beträgt in der Regel 5 bis 55 Grad Celsius laut Herstellerangaben, wobei von einer Luftfeuchtigkeit von 10 - 90% R.H. ausgegangen wird. Eine Lagertemperatur der Disketten von ca. 20 Grad Celsius sowie eine Betriebstemperatur des 1541 bei ungefähr gleicher Temperatur erscheint als angemessen.

Schmieren der Laufwerksmechanik

Wie bereits erwähnt, sollte man das Schmiermittel in *geringen* Mengen auftragen, so daß kein Öl auf die später eingelegte Diskette tropfen kann. Am besten trägt man das Öl mit einem Tropföler, einer Spritze mit Injektionskanüle oder einer kleinen Ölkanne auf. Das dünnflüssige Nähmaschinenöl wird in kleinen Tropfen an den beschriebenen Stellen aufgetragen.

Um die mechanischen Teile des Diskettenlaufwerks schmieren zu können, ist, wie bereits weiter vorne beschrieben, zu verfahren. Das Entfernen der Schrauben, die das Gehäuseunterteil mit dem eventuellen Blechchassis verbinden, ist nicht nötig.

Bevor man die Steckverbindungen der Digitalplatine trennt und die Befestigungsschrauben der Platine entfernt, sollten aber die Stecker gekennzeichnet werden. Ist die Digitalplatine abgehoben, sieht man die mechanischen Teile des Laufwerks.

Die Führungen des Diskettenverschlusses sind ebenfalls mit einem Tropfen Öl zu schmieren. Wird das Laufwerk verschlossen, wird die Trägerplatte des Andruckfilzes über die Verlängerung des Hebels angehoben. Der Andruckfilz wird angehoben, damit er nicht im Schreib- /Leseschlitz der Diskettenhülle hängen bleibt und ein Entnehmen der Diskette verhindert.

Damit auch das konische Andruckrad leicht mitläuft, sollte auf die Unterlegscheibe und Welle ein Tropfen Öl gegeben werden.

Geschwindigkeit des Diskettenantriebsmotors

Daß die 1541 richtig funktioniert ist u.a auch von der konstanten Drehzahl der Diskette abhängig. Der Zusammenhang zwischen dem Antriebsriemen und der Drehzahl des Laufwerks wurde bereits besprochen. Schlagen Sie, falls erforderlich, noch einmal weiter vorne nach, wie sich die Synchronisation beim Schreiben oder Lesen bei veränderter Drehzahl verhält.

Man konnte an Diskettenlaufwerken, die sich ca. 1 Jahr in Betrieb befanden, feststellen, daß die Geschwindigkeit der Diskette geringfügig niedriger ist (-1 ms), was jedoch in der Regel noch nicht zu R/W-Errors führt. Mit der Betriebsdauer der 1541 ist die Zeit gemeint, in der sich die Diskette bei Lese-/Schreibzugriffen dreht, und nicht die Einschaltdauer des Gerätes.

In der täglichen Praxis jedoch gibt es einige Anwender, die die Laufzeit (Betriebsdauer) von einer halben Stunde um ein Vielfaches überschreiten. Um die Geschwindigkeit zu überprüfen, sind die Vorbereitungen, wie bereits beschrieben, durchzufüh-

ren. Keine Angst, so schwer ist das nicht. Wichtig ist zunächst, daß Sie die bereits ergangenen Hinweise zur Sicherheit beachten und die besprochenen Grundprinzipien verstanden haben. Befindet sich das Chassis inklusive Platine und Laufwerk in senkrechter Lage, so daß der Einschubschlitz der Diskette nach vorne gerichtet ist, sehen Sie auf der Unterseite des Gerätes die Stroboskopscheibe.

In dem Raum, in dem Sie die Geschwindigkeit des Laufwerks (der Diskette) überprüfen, sollte sich eine Neonleuchte befinden. Diese Neonleuchte flimmert durch die Frequenz unseres Wechselstroms in Deutschland und anderen europäischen Ländern mit einer Frequenz von 50 Hertz. Das menschliche Auge ist zu träge, um das Wechseln des Stromes von 50 Hertz pro Sekunde zu erfassen. Damit sich die Stroboskopscheibe ausreichend lange dreht, ist eine Diskette zu formatieren und ein Programm zu laden oder abzuspeichern. Dreht sich die Stroboskopscheibe sollte die innere Skala, die mit "50" für 50 Hertz bezeichnet ist, ein stehendes Bild ergeben.

Bewegt sich das Bild der Stroboskopscheibe in Drehrichtung, läuft die Diskette zu schnell. Sollte es sich gegen die Drehrichtung bewegen, läuft das Laufwerk zu langsam. Ein sich nur geringfügig bewegendes Bild auf der Stroboskopscheibe ist noch kein Anlaß, die Geschwindigkeit zu justieren. Wie in diesem Unterkapitel bereits beschrieben, laufen die meisten Laufwerke nach einiger Zeit etwas langsamer, meist um -1 ms.

Die Diskette dreht sich mit 300 Umdrehungen in der Minute, d.h. 5 Umdrehungen pro Sekunde. Der Sollwert ist dabei 200 ms - Zeit = T, die sich aus einer Umdrehung ergeben.

9.2.5 Modifikation des Stopprings

Am Schrittmotor, der für den Transport des Schreib-/Lesekopfes zuständig ist, ist ein Ring mit zwei Anschlagflächen aufgepresst. Davon dient die eine Anschlagfläche des Stopprings als Endeanschlag der äußersten Position des Schreib-/Lesekopfes.

Sollte Ihr Stoppring keine Bohrung besitzen, kann ich Ihnen von einem nachträglichen Anbringen der Bohrung zur Sicherung des Stoppringes nur abraten. Also lassen Sie besser die Finger von irgendwelchen Versuchen, die Sie vielleicht sehr viel Geld kosten könnten.

Bei Laufwerken mit Stoppringen auf der Schrittmotorwelle ist ein Verkleben der Verbindung ratsam, wobei außerhalb des Schwenkbereichs die Verbindung mit Sicherungslack festzulegen ist. Die Fläche des Stoppringes ist der Schwenkbereich. Haben Sie keinen Sicherungslack oder ähnliches zur Hand, können Sie auch Panzerkleber oder Sekundenkleber für Metall verwenden.

Es sind 2 bis 3 Tropfen Sicherungslack auf die Verbindung der Welle und des Rades zu geben. Damit der Sicherungslack aushärten kann, sollte das Laufwerk mehrere Stunden nicht benutzt werden. Um diese Arbeiten durchführen zu können, ist die Demontage, wie schon beschrieben, durchzuführen und die Digitalplatine abzuschrauben. Dazu sind ebenfalls die Steckverbindungen zu markieren, damit kein Verpolen der Stecker bei der Montage der Platine vorkommt.

9.3 Laufwerkspraxis

Prüfen der Justagen mit dem Programmm Display T&S

Wie in Kapitel 9.1.1 bereits erwähnt, kann die Einstellung der Spur 1 mit dem Programm "DISPLAY T&S" überprüft werden.

9.3.1 Prüfen der Einstellungen - Soll-Ist-Vergleich

Laden Sie das Programm DISPLAY T&S mit LOAD, und starten Sie es mit <Run>. Daraufhin erscheint folgende Meldung:

```
Display Block Contents
Screen
or
Printer
```

Legen Sie erst eine nicht bespeicherte, formatierte Diskette ins Laufwerk, bevor Sie die Taste <S> für "Bildschirmausgabe" betätigen. Daraufhin erfolgt die Meldung

```
Track, Sector?
```

auf dem Bildschirm. Nun betätigen Sie die Taste <1> für die Spur-1-Überprüfung und die <Return>-Taste. Nun werden zwei Fragezeichen auf den Bildschirm gedruckt und die Eingabe der Sektorennummer erwartet. Geben Sie auch hier wieder die <1> ein und schließen Sie mit der <Return>-Taste die Eingabe ab.

Bei angeschlossenem Laufwerk und eingelegter Diskette fährt nun der Schreib-/Lesekopf auf die Spur 1 der Diskette. Wenn eine leere Diskette eingelegt ist, sind die Bytes des Sektors auf 0 gesetzt, mit Ausnahme des ersten Bytes, das EA anzeigt.

Es wird folgendes Bild auf den Bildschirm ausgegeben:

```
Track 1 Sector 1
00 :EA 00 00 00
04 :00 00 00 00
08 :00 00 00 00
```

bis

```
7c :00 00 00 00
Continue (Y/N)
```

Durch dieses Hilfsmittel können Sie sich alle 254 Bytes eines Sektors ansehen. Somit sehen Sie auch, ob sich der Schreib-/Lesekopf auf die Spur 1 bewegen kann und die Daten der Spur 1 gelesen werden.

Probieren Sie jetzt das Lesen von anderen Sektoren auf der Spur 1. Sie werden dabei feststellen, daß diese Sektoren dieselben Daten wie der Sektor 1 enthalten.

Sehen Sie sich auch andere Spuren und Sektoren auf der Diskette an, geben Sie dabei aber keine Spuren oder Sektoren an, die auf der Diskette nicht existieren. Belegung der Spuren:

Spur	Sektor-Nr.	Sektoren
1 - 17	0 - 20	21
18 - 24	0 - 18	19
25 - 30	0 - 17	18
31 - 35	0 - 16	17

In der Praxis mit der 1541 ist es auch schon vorgekommen, daß die Spuren 1 bis 33 oder 34 gelesen wurden und die Spur 35 nicht. Dieser Fehler ist auch auf eine Dejustage der physikalischen Spur 1 zurückzuführen, da bei der 1541 die Spuren von der Spur 1 nach innen bis zur Spur 35 gezählt werden.

Zur Überprüfung, daß die Spur 35 gelesen wird, sollten Sie sich einen Sektor dieser Spur mittels DISPLAY T&S ausdrucken lassen. Der Inhalt des Sektors bei einer leeren Diskette ist gleich mit dem der Spur 1. Sollte Ihnen das Programm DISPLAY T&S bisher unbekannt sein, dann nehmen Sie sich die Zeit, mit diesem Hilfsprogramm etwas zu experimentieren.

Besonders die Spur 18 ist sehr interessant, die ja bekanntlich vom Betriebssystem der 1541 beschlagnahmt wird.

Im Sektor 0 der Spur 18 befindet sich die BAM (Block Availability Map), zu deutsch "Blockverfügbarkeitstabelle". Hier wird festgehalten, welche Blöcke noch verfügbar sind, welche schon beschrieben sind, der Name der Diskette und die ID. Das Directory der Disk beginnt im Sektor 1 der Spur 18.

Wurde die Kontrolle der Spur 35 durchgeführt und keine Mängel festgestellt, sollten Sie trotzdem kritisch bleiben. Nehmen wir an, es liegt eine Dejustage des Spur-1-Anschlages vor, dann wird eine Diskette ja zur normalen Formatierungsposition versetzt formatiert.

Machen Sie sich klar, das ein Floppy so nicht funktionieren kann. Die Folge ist, daß Disketten, die vor der Verstellung des Laufwerks formatiert und bespielt wurden, nicht oder nur teilweise gelesen werden. Ebenso verhält es sich bei Programmen auf Diskette, die Sie aus Ihrem Bekanntenkreis bekommen. Zur

Sicherheit kann man diese Überprüfung mit einer Diskette vornehmen, die zu einem früheren Zeitpunkt formatiert und auch bespielt wurde.

Das hört sich etwas kompliziert an, ist aber sehr einfach. Es ist davon auszugehen, daß, wenn Sie eine Diskette verwenden, die auf einer intakten Floppy formatiert wurde, die logische Lage der Spur 1 auf der Diskette als Sollwert dienen kann, da bei einem richtig eingestellten Laufwerk die physikalische Lage der Spur 1 dem Sollwert entspricht.

Wenn die Überprüfung mit dieser Diskette und dem Programm DISPLAY T&S wiederholt wird, kann eine falsche Beurteilung des Laufwerks ausgeschlossen werden. So ist es möglich, einen echten Soll-Ist-Wertvergleich zu erstellen. Ist es dabei nicht möglich, die Spur 1 oder 35 mit DISPLAY T&S zu lesen, ist eine Justage der Spur 1 vorzunehmen.

Wichtige Hinweise

Bei dem Programm DISPLAY T&S handelt es sich quasi nur um einen Lesetest. Um eine Dejustage der Einstellungen Ihres Laufwerks zu erkennen, reicht in den meisten Fällen das Programm DISPLAY T&S aus, mit dem Sie alle Spuren einer Diskette lesen können.

Fehler, die nur im Schreibzugriff der 1541 vorkommen, treten selten auf und lassen in der Regel nicht auf eine Dejustage des Schreib-/Lesekopfes oder der Spur 1 schließen.

Ist eine dieser mechanischen Einstellungen oder sind beide verstellt, erhalten Sie auch READ-ERROR-MELDUNGEN und nicht nur WRITE-ERRORs. WRITE ERRORs lassen bei intaktem Leseverhalten Ihrer 1541 in der Regel auf einen Hardware-Fehler schließen, vor allem, wenn es auf allen Spuren keine Leseprobleme gibt und selbst Programmme oder Spiele mit 100 Blocks und mehr einwandfrei gelesen werden.

Wie bereits angesprochen, gibt es die Möglichkeit, daß der Schreib-/Lesekopf und der Stoppanschlag für die Spur 1 ver-

stellt sind. Hier stellt sich die Frage, welche der beiden Justagen wohl wichtiger ist. Die Antwort dazu ist sehr einfach: Die beiden Einstellungen besitzen die gleiche Priorität, wobei die Einstellung des W-K-Kopfes von der Einstellung des Stoppanschlags der Spur 1 abhängig ist.

Das heißt, es muß zuerst die Justage des Stoppanschlags der Spur 1 überprüft und - falls notwendig - eine Nachjustierung durchgeführt werden. Eine Nachjustierung am Stoppanschlag der Spur 1 zieht eine Neueinstellung des R/W-Kopfes mit sich. Deshalb sollten Sie sich nicht zu übereilten Eingriffen hinreißen lassen.

Ein Nachstellen des Schreib-/Lesekopfes hingegen erfordert nicht die Nachjustierung des Stoppanschlags der Spur 1, weshalb Sie aber nicht weniger kritisch handeln sollten. Führen Sie deshalb alle Ihnen zur Verfügung stehenden Tests kritisch durch, bevor Sie sich für eine Nachjustierung entscheiden.

Laden bei verstelltem Drive

Ist es Ihnen nicht mehr möglich, auch kurze Programme mit Ihrem Diskettenlaufwerk zu lesen, sollten Sie sich eine Datasette zur Hilfe nehmen. Bei stark verstelltem Laufwerk kommt es vor, daß bereits Programme mit wenigen Blöcken nicht gelesen werden. Trotzdem ist es in den meisten Fällen (99%) möglich, einen R/W-Test oder das Programm DISPLAY T&S zu betreiben. Nach folgender Reihenfolge ist zu verfahren, wenn Sie eine 1541 besitzen, die Ihnen das Laden der Testprogramme nicht ermöglicht.

Zweite 1541

Steht Ihnen eine zweite 1541 zur Verfügung, dann laden Sie den gewünschten Test mit diesem Gerät in Ihre Zentraleinheit. Daraufhin wird die 1541 ausgeschaltet und die beiden Stecker an der Geräterückseite des Diskettenlaufwerks werden abgezogen. Schließen Sie jetzt das zu testende Gerät an, schalten Sie es ein, und starten Sie den jeweiligen Test.

Datasette

Mit der Datasette können Sie Ihre Testprogramme laden und das gleichzeitig angeschlossene Diskettenlaufwerk testen.

Laufwerk oder Digitalplatine

Können Sie die genannten Tests mit Ihrer 1541 noch laden, sind die folgenden Zeilen für Sie weniger wichtig. Mußten Sie jedoch das Testprogramm mit einem zweiten Gerät laden, sollten Sie folgende Zeilen aufmerksam lesen.

Sollte Ihre 1541 nach dem Starten des jeweiligen Testprogramms keine Reaktion zeigen, ist es möglich, daß ein elektronischer Fehler vorliegt.

Überzeugen Sie sich davon, daß Ihr Commodore 64 keinen Defekt aufweist. Es kann durchaus sein, daß die serielle-IEEE-488-Schnittstelle defekt ist. Überprüfen Sie Ihren Commodore 64, indem Sie Ihren Rechner an ein intaktes Diskettenlaufwerk anschließen. Lädt Ihr 64'er auch von diesem Gerät nicht, ist mit Sicherheit die serielle Schnittstelle des Commodore 64 oder auch eine andere Komponente defekt. In der Regel erhalten Sie dann beim Laden des Directory folgenden Ausdruck:

```
LOAD "$",8
SEARCHING FOR "$"
xxxxxxxx  : Die Meldung LOADING erscheint bei
          : intaktem Gerät an dieser Stelle.
          : Bei einem defekten jedoch nicht.
READY.
```

Erhalten Sie beim Laden des Inhaltsverzeichnisses folgende Sequenz, ist der Fehler bei Ihrer 1541 zu suchen:

```
LOAD "$",8
Searching for $  : Nach dieser Meldung stürzt Ihr Commodore
                 : 64 ab (der Cursor erscheint nicht mehr),
                 : und es ist keine weitere Eingabe
                 : mehr möglich.
```

Weiterhin sollten Sie darauf achten, daß nach dem Einschalten die Leuchtdioden (LED) in der üblichen Reihenfolge aufleuch-

ten. Ist dies nicht der Fall und dreht sich der Motor des Laufwerks, seitdem Sie eingeschaltet haben, dann erhält Ihre Diskettenstation ein ständiges RESET-Signal.

Dieser RESET ist Ihnen wahrscheinlich schon von Ihrem 64'er durch den "RESET-Taster" bekannt. Deshalb kann es sein, daß die Ihnen zur Verfügung stehenden Tests nicht ausgeführt werden, nachdem sie mittels eines zweiten Gerätes geladen wurden.

9.3.2 Geschwindigkeit justieren

Schalten Sie Ihre Anlage aus, und ziehen Sie danach die Stecker aus den Gerätebuchsen. Vorbereitungen zur Justage oder Reparaturarbeiten sollten Sie stets bei stromlosem Gerät durchführen. So wird das Risiko einer ungewollten Beschädigung vermindert.

Um die Geschwindigkeit nachstellen zu können, ist die Demontage des Gehäuses, wie bereits beschrieben, durchzuführen. Zur Durchführung der Justage ist nur die Steckverbindung zu der in der Gehäusefront befindlichen Leuchtdiode abzuziehen. Nachdem Sie das Chassis in die richtige Position gebracht haben, können Sie Ihre 1541 wieder anschließen und einschalten. Wie Sie wissen, ist die Elektronik der 1541 gegen elektrostatische Aufladungen sehr empfindlich, vor allem sind es die großen Bausteine auf der Digitalplatine, die bei elekrostatischer Entladung beschädigt werden können.

Mit dem Programm "Geschwindigkeitsjustage" ist das genaue Einstellen des Diskettenantriebsmotors möglich. Dieses Programm liegt als Listing vor. Nachdem Sie das Programm in den Rechner eingetippt haben, ist eine formatierte Diskette in das Laufwerk einzulegen und zu starten. Geschwindigkeitsabweichungen werden durch -1 ms, -2 ms oder +1 ms oder mehr angezeigt.

```
10 POKE53281,1:POKE53280,1:PRINTCHR$(30)
20 PRINTCHR$(147)
30 OPEN1,8,15
40 PRINT"    "CHR$(18)"GESCHWINDIGKEITSEINSTELLUNG"
50 PRINT
60 PRINTTAB(9)"BITTE WARTEN"
```

```
70 FOR I= 0 TO 161
80 READA:S=S+A
90 PRINT#1,"M-W";CHR$(I)CHR$(3);CHR$(1)CHR$(A)
100 NEXT I
110 IF S <> 15669 THENPRINT"FEHLER IN DATAS":GOTO480
120 PRINT
130 PRINT
140 PRINTTAB(4)"LEGEN SIE EINE UNFORMATIERTE  "
150 PRINT
160 PRINTTAB(10)"DISKETTE EIN"
170 GOSUB560
180 PRINT
190 PRINT
200 PRINTTAB(4)"WENN FERTIG "CHR$(18)"SPACE"CHR$(146)" DRUECKEN"
210 GETA$:IF A$ <> CHR$(32) THEN210
220 PRINTCHR$(19)
230 FORI=0TO5:PRINTCHR$(17):NEXT I
240 PRINT
250 PRINT
260 PRINT
270 PRINT
280 PRINT
290 PRINT"DRUECKE "CHR$(18)"'F7'"CHR$(146)" WENN GESCHWINDIGKEIT
O.K. "
300 FORI=0TO2:PRINTCHR$(145);:NEXT
310 PRINT#1,"M-W"CHR$(5)CHR$(3)CHR$(1)CHR$(11)
320 T=36:J=14:GOSUB570
330 FORI=1TO4
340 PRINT#1,"M-R"CHR$(6+I)CHR$(3):GET#1,B$
350 S(I)=ASC(B$+CHR$(0))
360 NEXT I
370 IF S(3)=0 OR S(4)=0 THEN500
380 C=256*(S(4)+S(3))+S(2)+S(1)-3996
390 C=INT(C/20+0.5)
400 PRINTCHR$(145)"GESCHWINDIGKEITSABWEICHUNG IST "C CHR$(148)"MS
"
410 GETA$:IF A$= CHR$(136) THEN450
420 IF ABS(C) > 5 THEN310
430 PRINT#1,"M-W"CHR$(5)CHR$(3)CHR$(1)CHR$(78)
440 GOTO320
450 PRINTCHR$(17)CHR$(17)CHR$(17)
460 PRINT"O.K."
470 GOSUB560
480 CLOSE1
490 END
500 PRINT
510 PRINTCHR$(18)"FEHLER !!! ->"
520 PRINT"UEBERPRUEFEN SIE IHRE DISKETTE!!!     "
530 FORI=1TO1000:NEXT I
540 GOTO470
550 REM
560 T=1:J=12
570 PRINT#1,"M-W"CHR$(6)CHR$(0)CHR$(2)CHR$(T)CHR$(0)
```

```
580 PRINT#1,"M-W"CHR$(0)CHR$(0)CHR$(1)CHR$(J*16)
590 PRINT#1,"M-
R"CHR$(0)CHR$(0):GET#1,L$:L=ASC(L$+CHR$(0)):IFL>127THEN590
600 IF J=12 THEN FORI=1TO2500:NEXT
610 RETURN
620 REM
630 DATA169, 0,133, 0, 76, 11, 3, 0
640 DATA 0, 0, 0,120,173, 12, 28, 41
650 DATA 31, 9,192,141, 12, 28,169,255
660 DATA141, 3, 28,162, 85,142, 1, 28
670 DATA162, 50,160, 0, 80,254,184,136
680 DATA208,250,202,208,247, 80,254,184
690 DATA141, 1, 28,169,224, 13, 12, 28
700 DATA162, 4, 80,254,184,202,208,250
710 DATA141, 12, 28,142, 3, 28,162, 3
720 DATA 80,254,184,202,208,250,120,173
730 DATA 11, 24, 9, 64,141, 11, 24,162
740 DATA 1,169, 98,141, 4, 24,160, 0
750 DATA140, 8, 3,140, 10, 3,140, 7
760 DATA 3,140, 9, 3, 44, 0, 28, 48
770 DATA251,140, 5, 24, 44, 0, 28, 16
780 DATA251, 44, 0, 28, 16, 19,173, 13
790 DATA 24, 10, 16,245,173, 4, 24,254
800 DATA 7, 3,208,237,254, 9, 3,208
810 DATA232,202,240,224,169,191, 45, 11
820 DATA 24,141, 11, 24,169, 1, 88,108
830 DATA232,255
```

Um einen stroboskopischen Effekt zu erhalten, benötigen Sie in Ihrem Arbeitszimmer eine Neonleuchte, mit der es Ihnen möglich ist, anhand der Stroboskopscheibe die Geschwindigkeit des Laufwerks einzustellen. Ihr Diskettenlaufwerk sollte senkrecht auf einer der beiden Seitenflächen des Chassis stehen. Steht das Chassis des Gerätes so, daß die rote Leuchtdiode des Laufwerks oben ist, und blicken Sie nun von links auf die Geräteunterseite, sehen Sie in der linken unteren Ecke des Chassis die Bohrung, hinter der sich das Potentiometer befindet.

Der veränderbare Widerstand befindet sich auf einer kleinen Platine, die zur Ansteuerung der beiden Laufwerksmotoren dient. Diese Platine ist meist an die Unterseite des Laufwerks geschraubt.

Wie bereits erwähnt, befindet sich auf der Unterseite des Blechchassis das Potentiometer. Sinnbildlich im Sinne eines Uhrenziffernblattes gesehen, steht in der Regel der Schlitz des ca.

auf 11 Uhr und 30 Minuten, wobei die Seite, die näher zur Gehäuseaußenseite liegt, mit 12 Uhr bezeichnet wird. Zum Einstellen der Geschwindigkeit ist es empfehlenswert, eine Diskette zu formatieren oder ein sehr langes Programm zu laden. So haben Sie ausreichend Zeit, auf die Stroboskopscheibe zu sehen, und die innere Markierung, die mit "50" für 50 Hertz bezeichnet ist, für das Einstellen des Diskettenantriebsmotors zu beobachten. Erhalten Sie mit der inneren Markierung der Stroboskopscheibe ein stehendes Bild, so ist das Nachjustieren Ihres Drives nicht nötig.

Dreht sich das Bild der 50-Hertz-Stroboskopscheibe gegen die Drehrichtung der Aluminiumscheibe, dann läuft der Motor Ihres Laufwerks zu schnell. Umgekehrt bewegt sich bei zu langsam drehendem Motor die Markierung in die Drehrichtung der Antriebsscheibe, also links herum.

Stellen Sie nun die Geschwindigkeit des Antriebsmotors mit einem kleinen Schraubendreher am Widerstand ein. Beobachten Sie nun die 50-Hertz-Skala der Stroboskopscheibe, und drehen Sie ein wenig in Richtung 12 Uhr. So läuft der Diskettenantriebsmotor schneller. Verstellen Sie das Potentiometer in Richtung 11 Uhr, so läuft der Antriebsmotor langsamer. Sollten Sie zuviel nach rechts gedreht haben, dreht sich die 50-Hertz-Stroboskopscheibe gegen die Drehrichtung der Antriebsscheibe.

Drehen Sie auf keinen Fall unnötig eine viertel Umdrehung nach rechts oder links! Bei dieser Justage ist etwas Feingefühl notwendig. Dreht das Bild mit der Drehrichtung der Antriebsscheibe, so läuft der Antriebsmotor zu langsam. Sie sollten auf keinen Fall mit Ihrer 1541 experimentieren, da dieses Gerät als Elektronik-Experimentierkasten zu wertvoll ist.

Bei der Durchführung der Geschwindigkeitseinstellung werden Sie kaum ein vollkommen stehendes Bild der Stroboskopscheibe erhalten. Selbst, wenn Sie die Justage sehr genau durchführen, werden Sie bemerken, daß sich das Stroboskopbild geringfügig bewegt. Nachdem die Justage der Geschwindigkeit beendet ist, sollten Sie das Potentiometer mit einem Tropfen Nagellack sichern, damit es sich nicht von selbst verstellen kann. Nachdem

Sie die erste Justage an Ihrem Diskettenlaufwerk der 1541 vorgenommen haben, werden Sie feststellen, daß es nicht allzu schwierig ist, irgendwelche Justagen durchzuführen.

9.3.3 Lage der Spuren 1-35 einstellen

Die Justage des Stoppanschlags für die Spur 1 auf der Diskette können Sie nach Ablauf der Garantiezeit durchaus selbst erledigen. Ein Verstellen der Spur-1-Justierung stellt man kaum von der einen auf die andere Minute fest.

Aber auf einmal merkt man doch, daß das eine oder andere Programm nicht geladen wird. Dann sollten Sie, wie es schon weiter vorne besprochen wurde, anhand der Testprogramme Ihr Laufwerk überprüfen. Nur, wenn Sie anhand der Ihnen zur Verfügung stehenden Testprogramme eine Dejustage des Stoppanschlags für die Spur 1 auf der Diskette festgestellt haben, sollten Sie zum Schraubendreher greifen.

Es ist nicht besonders wichtig, gut mit dem Schraubendreher umgehen zu können, zu dieser Justage sind Aufmerksamkeit und Feingefühl die wesentlichen Eigenschaften. Das sorgfältige Überprüfen mit den Programmen DISPLAY-T&S oder PERFORMANCE TEST während der Einstellarbeiten ist eine Voraussetzung.

Demontieren Sie das Gehäuse Ihrer 1541, und führen Sie die nötigen Vorbereitungen durch. Sie sollten Demontage und Vorbereitungsarbeiten nur bei *stromlosem* Gerät durchführen. Dadurch vermeiden Sie Kurzschlüsse, die durch eine kleine Unachtsamkeit hervorgerufen werden können. Eine Beschädigung der Digitalplatine durch Kurzschluß oder ähnlichem ist schwieriger zu beseitigen als die Dejustierung eines Laufwerks.

Einstellen des Stoppanschlags für die Spur 1

Der Stoppring mit seinen zwei Anschlagflächen dient als Endeanschlag für die Spur 1. Der weiße Teil des Stopprings ist der mögliche Arbeitsbereich des R/W-Kopfes.

Initialisieren Sie Ihre Diskettenstation mit dem folgendem Befehl, dann fährt der Schreib-/Lesekopf nach außen, schlägt mit dem Stoppring gegen den Endeanschlag, und von da an zählt Ihre 1541 17 Spuren nach innen, bis der R/W-Kopf auf Spur 18 der Diskette steht. Entnehmen Sie dazu zuerst die Diskette, damit Sie den R/W-Kopf beobachten können.

```
OPEN 15,8,15,"I"
```

Das Initialisieren ist auch bei dejustiertem Laufwerk möglich und nicht von der richtigen Einstellung des Stoppanschlags abhängig. Nehmen wir an, der Stoppanschlag ist verstellt und steht mit der Vorderkante zu nahe an der Welle des Schrittmotors. Auf dieser ist der Stoppanschlagring befestigt.

Der Stoppanschlagring ist über das Metallband mit dem R/W-Kopf verbunden und transportiert somit den Schreib-/ Lesekopf des Laufwerks. Sollen nun Daten bei obengenannter Dejustierung von der Spur 1 einer Diskette geladen werden, die mit einem intakten Laufwerk formatiert und bespielt wurde, ist dies nicht möglich. Daraufhin blinkt die rote Leuchtdiode Ihrer 1541 und signalisiert eine Fehlermeldung. Bei der Abfrage der Fehlermeldung erhalten Sie die Meldung: 20 : READ ERROR. Der Disk-Controller kann den Header (Vorsatz) eines Datenblocks nicht finden. Abfragen des Fehlerkanals:

```
10 OPEN 1,8,15
20 INPUT#1,A,B$,C,D
30 PRINT A,B$,C,D
```

Wenn der Stoppanschlag zu weit innen bzw. zu nahe am Stoppring steht wird die Spur 1 vom R/W-Kopf nicht erreicht. Zur Nachjustierung des Stoppanschlags 2 benötigen Sie das Programm DISPLAY T&S oder R/W-Test und die bereits erwähnten Arbeitsdisketten. Laden und starten Sie DISPLAY T&S wie, es bereits ausführlich besprochen wurde. Kann die Spur 1 einer Diskette, die auf einem intakten Laufwerk formatiert oder beschrieben wurde, nicht gelesen werden ist folgendes zu tun:

Lösen Sie die Kreuzschlitzschraube mit einem passenden Schraubendreher. Nun vergrößern Sie mit einem normalen Schrauben-

dreher den Abstand des Stoppanschlags und der Welle des R/W-Kopftransportmotors, indem Sie den Schraubendreher zwischen die Welle und das Anschlagblech setzen und das Stoppanschlagblech von der Welle weg um ca. 0.5 mm nach außen drücken. Drehen Sie daraufhin die Schraube fest, und überprüfen Sie die Justage, wie es in Unterkapitel 9.3.2 bereits besprochen wurde.

Wenn jetzt die Spur 1 der Diskette gelesen wird, sollte der Abstand zwischen Stoppanschlag und Stoppring ca. 0.35 mm betragen. Gegebenenfalls ist die Einstellung des Stoppanschlags zu korrigieren, bis die Spur 1 der Arbeitsdiskette gelesen wird. Der Abstand von 0.35 mm muß nicht eingehalten werden. Wichtiger ist die Funktion des Laufwerks. Nachdem Sie in diesem Fall den Abstand zwischen Stoppanschlag und Stoppring um 0.1 bis 0.5 mm vergrößert und mit der Schraube gesichert haben, ist der Stoppanschlag festzukleben. Dazu ist die Verbindung des Stoppanschlags und der Schraube mit einem Tropfen Kleber oder ähnlichem zu sichern. Dabei ist auch ein Tropfen Kleber in die Einfräsung des rechteckigen Aluminiumzapfens zu geben, auf dem der Stoppanschlag befestigt ist. Vom Hersteller wurde diese Verbindung ebenso gesichert.

Nach dem ersten Erfolg bei Ihrer Selbsthilfe werden Sie feststellen, daß dies nicht besonders schwierig war. Sollte Ihre 1541 noch nicht vollkommen funktionieren, ist das kein Anlaß aufzugeben. Ein Nachjustieren des Schreib-/Lesekopfes Ihrer 1541 muß nicht unbedingt erforderlich sein. Überprüfen Sie deshalb Ihr Laufwerk, wie es in Unterkapitel 9.3.2 bereits beschrieben wurde, und laden Sie außerdem mehrere lange Programme. Ist Ihr Laufwerk nun wieder in Ordnung und haben Sie alle Funktionen überprüft, ist die Montage des Gerätes durchzuführen. Dabei sollten Sie ebenfalls sorgfältig vorgehen.

Lage der Spuren 1 bis 35 einstellen

`Fehlermeldung: 20 Read Error - Spur 33 (34,35)`

Wird bei Ihrem Laufwerk die Spur 35 der Diskette nicht gelesen, wie es in Kapitel 9.3.2 bereits erwähnt wurde, ist der Stoppanschlag eventuell nachzustellen. Jedoch ist der Stoppanschlag in Richtung Stoppring zu verstellen. Bei dieser Dejustierung erhal-

ten Sie READ-ERROR-Meldungen in der Spur 35, gegebenenfalls auch schon ab der Spur 33. Zum Justieren des Laufwerks sind Disketten zu verwenden, die auf einem intakten Laufwerk formatiert wurden.

Steht bei Ihrem Diskettenlaufwerk der Stoppanschlag um 0,5 bis 1 mm zu weit außen, ist es möglich, daß die Spuren ab 32 bis 35 nicht gelesen werden.

Zum Nachstellen des Stoppanschlags benötigen Sie das Programm DISPLAY T&S und Disketten, die mit einer intakten 1541 formatiert wurden. Überprüfen Sie das Laufwerk mit dem Ihnen zur Verfügung stehenden Programm, wie es in Unterkapitel 9.3.2 bereits besprochen wurde. Positionieren Sie den R/W-Kopf mit dem Programm DISPLAY T&S auf Spur 1, wie es in Unterkapitel 9.3.2 beschrieben ist. Lösen Sie nun die Schraube mit einem passenden Schraubendreher. Der Stoppanschlag ist so zu verstellen, daß der Luftspalt zwischen der Anschlagfläche des Stopprings und dem Stoppanschlag geringer wird.

Überprüfen Sie die Justage mit Ihren verschiedenen Testprogrammen, und laden Sie mehrere lange Programme oder Spiele, die auf den Spuren 30 bis 35 abgespeichert sind. Sichern Sie die Schraube und den Stoppanschlag mit Kleber oder Nagellack. Auch hier ist nicht der Abstand von 0,35 mm zwischen Stoppanschlag und Stoppring entscheidend, sondern die Funktion des Laufwerks. Im Werk des Herstellers wird der Abstand zwischen Stoppanschlag und Stoppring auf 0,35 mm eingestellt, wobei sich der Schreib-/Lesekopf auf der Spur 1 befindet. Betreiben Sie Ihre 1541 längere Zeit, ist es möglich, daß sich das eine oder andere Teil Ihres Laufwerks abnutzt oder daß sich Grundeinstellungen des Laufwerks verändert haben. In der Regel ist dies aber selten der Fall.

Dadurch kann es beim Nachjustieren Ihres Laufwerks erforderlich sein, daß Sie den Abstand von 0,35 mm nicht einhalten können. Es kann sein, daß Sie den Stoppanschlag so weit verstellen müssen, daß der Abstand nur noch 0,1 mm beträgt. Mußte der Stoppanschlag stark verstellt werden, um die eine

oder andere Spur wieder lesen zu können, ist in der Regel ein Nachjustieren des Schreib-/Lesekopfes erforderlich.

9.3.4 Schreib-/Lesekopf einstellen

Einführung

Das Einstellen des Schreib-/Lesekopfes wird im Fachjargon "Alignment" genannt. Darunter versteht man die Justierung der radialen Schreib-/Lesekopfposition. In folgenden Schritten erhalten Sie eine Einführung zur Justage des Diskettenlaufwerks 1541. In einer Fachwerkstatt werden mit Hilfe eines Oszilloskops und einer Spezialdiskette die Amplitude des Schreib-/Lesekopfes gemessen. Auf dieser Spezialdiskette (Alignment-Diskette genannt) befindet sich auf der Spur 17 ein Bit-Muster zum genauen Einstellen des Schreib-/Lesekopfes. Schließt man am Leseverstärker der Diskettenstation 1541 ein geeignetes Oszilloskop an, erhält man zwei Amplitudenschwingungen auf dem Bildschirm des Oszilloskops.

Diese beiden Amplitudenschwingungen haben die Form zweier Katzenaugen und werden deshalb als "Cats Eyes" in der Fachsprache bezeichnet. Bei einer Ablenkzeit von 20 msec/div des Oszilloskopes sind auf dem Bildschirm zwei gleiche Cats Eyes sichtbar, welche bei korrekter Einstellung eine Umdrehung der Alignment-Diskette darstellen. Als Amplitude wird die Signalhöhe eines Impulses bezeichnet. Die Signalhöhe wird in der Einheit Volt gemessen. Die Bezeichnung 20 ms/div bedeutet, daß ein Kästchen auf dem Bildschirm eine Zeit von 20 Millisekunden darstellt. Der Bildschirm eines Oszilloskopes hat in der Regel 10 Kästchen in der Breite (horizontal), daraus ergeben sich 10 * 20 msec = 200 Millisekunden oder 0,2 Sekunden. Wie bereits erwähnt wurde, drehen sich die Disketten mit 300 Umdrehungen in der Minute. Daraus ergeben sich 300 U. pro min / 60 sec = 5 U. pro sec, und das ergibt wiederum 1 sec / 5 U. pro sec = 0,2 Sekunden für eine Umdrehung der Diskette. So ist es möglich, die Spur 17 der Alignment-Diskette auf dem Bildschirm des Oszilloskopes darzustellen.

Eine Alignment-Diskette läßt sich nicht kopieren und ist für Ihre 1541 im Fachhandel für ungefähr 200 DM erhältlich. Diese Diskette ist keine Programmdiskette, sondern eine Meßdiskette. Bei der Justierung der radialen Schreib-/Lesekopfposition handelt es sich um eine Feineinstellung.

Ist der Schreib-/Lesekopf exakt abgeglichen, sind die beiden Cats Eyes gleich groß. Ist das erste Katzenauge auf dem Bildschirm um 25 % kleiner als das zweite, liegt eine Fehljustierung von 0,05 mm in Richtung Spur 16 vor. Wenn das zweite Katzenauge auf dem Oszilloskopschirm um 25 % kleiner ist als das erste, liegt eine Fehljustierung von 0,05 mm in Richtung Spur 18 vor. Eine Fehljustierung von 0,05 mm macht sich noch nicht bemerkbar. Erst, wenn das eine Katzenauge weniger als 75 % der Amplitude des anderen hat, macht sich eine Dejustierung durch READ oder WRITE ERRORs bemerkbar.

Können mit Ihrem Diskettenlaufwerk nur noch kürzere Programme eingelesen werden, dann ist in der Regel der Schreib-/Lesekopf sehr stark verstellt, und es wären keine Katzenaugen auf dem Bildschirm sichtbar. Eine solche Dejustierung kann durch äußere Einflüsse hervorgerufen werden, z.B. wenn das Diskettenlaufwerk beim Transport heftig gestoßen wird oder wenn es beim Versand fällt. In diesem Fall steht der Schreib-/Lesekopf nicht auf der Alignment-Spur 17 der Alignment-Diskette, sondern auf Spur 16 oder vielleicht sogar auf Spur 18. Dann ist es möglich, daß der Schrittmotor des R/W-Kopfes um einige Millimeter verstellt ist und neu justiert werden muß. Der Schreib-/Lesekopf wird nachgestellt, indem man die Schrauben löst, danach den Schrittmotor in die eine oder andere Richtung verdreht und dabei den Feinabgleich des Schreib-/Lesekopfes vornimmt.

Vorbereitungen zur Justage des Schreib-/Lesekopfes

Nachdem Sie im letzten Unterkapitel die Grundlagen für das Justieren des Schreib-/Lesekopfes erfahren haben, wollen wir diese nun in die Praxis umsetzen. Sicher haben Sie sich die Frage gestellt, wie es Ihnen möglich ist, ohne Oszilloskop die Justierung des Schreib-/Lesekopfes durchzuführen.

Justierung des Schreib-/Lesekopfes ohne Oszilloskop

Es braucht wohl nicht darauf hingewiesen zu werden, daß das folgende Unterkapitel nur für den sinnvoll ist, der weiß, an welchem Ende ein Lötkolben heiß wird. Sollten Sie sprichwörtlich zwei linke Hände besitzen, ist es für Sie ratsam eine Commodore-Fachwerkstatt aufzusuchen. Nachdem Sie diese Warnung erhalten haben, sollen Sie nicht davon abgehalten werden, Ihr Laufwerk selbst zu justieren, zumal dies mit etwas Feingefühl und Aufmerksamkeit kein Problem ist.

Vorbereitungen

Ist es Ihnen nicht möglich, mit Ihrer Diskettenstation lange Programme zu laden, und erhalten Sie daraufhin Fehlermeldungen von 20 - 24 ist es sehr wahrscheinlich das der Schreib-Lese Kopf verstellt ist.

Werden bei den Programmen PERFORMANCE TEST oder R/W-Test in mehreren Spuren READ ERRORs gemeldet, ist das Laufwerk in der Regel stark dejustiert. Im Unterkapitel 9.3.2 wurden die Möglichkeiten zur Überprüfung eines Laufwerks bereits ausführlich besprochen. Sind Sie sich noch nicht sicher, ob eine Dejustage des Schreib-/Lesekopfes vorliegt, sollten Sie Unterkapitel 9.3.2 wiederholen. Auch Kopierprogramme, wie Quickcopy oder ähnliche, geben häufig Aufschluß über das Schreib-/Leseverhalten eines Laufwerks.

Die einzige aufschlußreiche Möglichkeit, um festzustellen, ob eine Dejustage des Schreib-/Lesekopfes vorliegt, ist, das Lesesignal am Leseverstärker der 1541 nachzumessen.

Erfolgt die Justage des Schreib-/Lesekopfes im Anschluß zur Spur-1-Justage, müssen keine weiteren Vorbereitungen bzw. Demontagen durchgeführt werden. Das Laufwerk ist nur noch so auf eine Seitenfläche zu stellen, daß es senkrecht auf der Unterlage steht. In dieser Position sind die Befestigungsschrauben des Schrittmotors, der für den Schreib-/Lesekopftransport verantwortlich ist, gut zugänglich.

Demontagen zur Justierung des Schreib-/Lesekopfes

Ist nur die Justierung des Schreib-/Lesekopfes erforderlich sind die Vorbereitungen aus Unterkapitel 9.2.2 auszuführen. Nach Ausführung der Vorbereitungen sehen Sie die Befestigungsschrauben, die der Befestigung des exentrisch gelagerten Schrittmotors dienen.

Schalten Sie die Diskettenstation aus, und ziehen Sie den Stecker der 220-Volt-Zuführung ab. Die nötigen Vorbereitungen zur Justierung des Schreib-Lesekopfes sollten stets bei stromlosen Gerät durchgeführt werden. Dadurch vermeiden Sie eine Beschädigung der Digitalplatine, die durch eine Unachtsamkeit hervorgerufen werden kann. Nachdem die Vorbereitungen aus Unterkapitel 9.2.2 durchgeführt wurden, sehen Sie auf der Unterseite des Blechchassis die Aussparungen und die Schrauben. Der Schrittmotor ist mit einer abgesetzten Welle in das Aluminiumchassis des Laufwerks eingesetzt. Mit den beiden Blechlaschen des Motors und den Schrauben wird der Motor befestigt. Wird eine der vier Wicklungen des Schrittmotors bestromt, wird die Welle und der damit verbundene Stoppring um 1,8 Grad gedreht und somit der mit dem Metallbändchen verbundene Schreib-/Lesekopf transportiert.

Durch Verdrehen des Schreib-/Lesekopftransportmotors ist es möglich, die Justage des R/W-Kopfes durchzuführen.

9.4 C64-Wartung

Das Ziel dieses Kapitels ist es, Ihren C64 im bescheidenen Maßstab aufzurüsten und - im Falle eines Fehlers - diesen selbst zu lokalisieren und ihn gegebenenfalls zu reparieren.

Falls die Garantie Ihres Geräts noch nicht abgelaufen ist, sollten sie es sich vorher gut überlegen, ob Sie Ihr Gerät aufschrauben und damit Ihren Garantieanspruch verlieren.

Bevor Sie nach den in diesem Kapitel gegebenen Kurzanleitungen Geräte aufschrauben, ICs austauschen oder Sonstiges unter-

nehmen, untersuchen Sie als erstes, ob Sie wirklich alle nötigen Geräte ordnungsgemäß angeschlossen und eingeschaltet haben. Wenn sich bei einem Gerät überhaupt nichts tut, kontrollieren Sie als erstes die Kabel und Sicherungen. Nehmen Sie auch keine Eingriffe vor, die Sie sich nicht selbst zutrauen oder die Ihren Erfahrungsbereich weit übersteigen. Überschätzen Sie sich nicht, denn das kann Sie weitaus mehr kosten als die Reparatur bei einem Fachmann. In den folgenden Absätzen werden wir auf die häufigsten Fehlertypen näher eingehen:

Der Bildausfall

Unter "Bildausfall" verstehen wir, daß nach dem Einschalten des Computers und des entsprechenden Datensichtgeräts (Monitor, Fernseher) kein Bild erscheint, obwohl die LED am Computer leuchtet.

Falls dies der Fall ist, so ist mit größter Warscheinlichkeit ein Teil der Stromzuleitung unterbrochen, welches meistens auf eine defekte Sicherung zurückzuführen ist. Um dies festzustellen, lösen Sie die drei Schrauben an der Unterseite des Computers und klappen den oberen Teil vorsichtig nach hinten weg. Den Standort der Sicherung können Sie in den nachfolgenden Bildern ermitteln (die Stärke der Sicherung ist entweder 1 oder 1.25 Ampere). Anschließend nehmen Sie die Sicherung vorsichtig aus der Fassung und ersetzen sie gegebenenfalls durch eine 1.25 Ampere tragende Sicherung. Es ist häufig der Fall, daß die Sicherung ohne besonderen Grund durchbrennt. Der Ersatz durch eine 1.25 Ampere starke Sicherung ist völlig ohne Risiko, dagegen kann eine Überbrückung der Sicherung durch ein Stück Draht zu großen Schäden führen.

Ist die Sicherung in Ordnung, liegt der Fehler wahrscheinlich beim Transformator. Falls Sie über das entsprechende Meßgerät verfügen, können Sie in der DIN-Buchse nachmessen. Oft fehlt die 5-Volt-Gleichspannung, die von der Stabilisierungsschaltung im Netzteil geliefert werden soll. Aber auch die 9-Volt-Wechselspannung ist vereinzelt nicht vorhanden.

Wenn Sie allerdings nicht die Möglichkeit haben, das Netzteil durchzumessen, tauschen Sie Ihr Netzteil durch das eines Freundes aus. Sollte sich herausstellen, daß der Fehler im Netzteil lag, so bringen Sie dieses zu Ihrem Fachhändler. Sollte sich herausstellen, daß auch das Netzteil funktioniert, so überprüfen Sie ihr Anschlußkabel an dem Monitor oder Fernseher. Sind die Kabel in Ordnung, so könnte der Modulator defekt sein. Um dieses zu überprüfen, bedarf es eines kleinen Tricks:

Schließen sie Ihre Floppy oder Ihre Datasette an den Computer an. Im Falle der Datasette drücken Sie <Shift>+<Run/Stop> und daraufhin die <Play>-Taste. Beginnt der Motor zu laufen, so wird der Modulator defekt sein, und Ihr Computer muß in die Werkstatt. Ist eine Floppy angeschlossen, so tippen Sie blind LOAD"$",8. Beginnt der Laufwerksmotor sich zu drehen, so gilt das oben gesagte.

Wenn nicht einmal die LED an Ihrem Computer leuchtet, so sollten Sie die Sicherung an Ihrem Netzteil überprüfen und gegebenenfalls durch eine gleicher Stärke ersetzen. Es ist auch möglich, daß Ihre Sicherung sich nur gelockert hat. Trifft beides nicht zu, so sollten Sie den Stecker für die Steckdose überprüfen.

Nur Bildschirm und Rahmen

Häufiger aber auch schwerwiegender sind die Fehler, wenn nach dem Einschalten der Rahmen und die Hintergrundfarbe erscheinen, danach jedoch jede Aktivität stoppt. In diesem Fall sollten Sie zuerst einmal die IRQ-Leitung messen. Im Normalfall, also bei funktionierendem Gerät, tritt an diesem Anschluß (Pin 3 des Prozessors) alle 16 Millisekunden ein negativer Impuls von ca. 200 Microsekunden Dauer auf. Mit einem Vielfachmeßgerät ist nur eine Gleichspannung von ca. 4.5 Volt zu messen. Mit einer LED und einem Widerstand von 200 Ohm kann man oben sehen, ob die Interrupts ausgelöst werden. Dazu wird der Widerstand mit 5 Volt verbunden und als Vorwiderstand für die LED benutzt. Die Kathode der LED wird an Pin 3 des Prozessors gehalten. Jetzt muß die LED soeben sichtbar leuchten und etwas flackern. Besser geeignet ist natürlich ein Logik-Tester oder ein Oszilloskop.

Was aber, wenn die Interrupts ausbleiben? Wenn die IRQ-Leitung nach dem Einschalten High ist, dann nach kurzer Zeit Low wird, kommen als Fehlerquellen das Betriebssystem-ROM, das RAM, die CIAs oder der Prozessor in Frage. Ein Fall für eine gut ausgerüstete Werkstatt, wenn Sie nicht die Möglichkeit haben, die ICs aus einem anderen Rechner zu probieren.

Sind die CIAs defekt, so ist dieser Fehler leicht festzustellen. Der C64 kann ohne CIA 2 arbeiten, und deshalb ist es möglich, die beiden ICs, deren Lage anhand der Bilder ermittelt werden kann, auszutauschen. Falls der Computer nun wieder funktioniert, müssen Sie das defekte IC, das vorher im Sockel von CIA 1 steckte, austauschen.

Hat diese Operation auch nicht den gewünschten Erfolg, können Sie untersuchen, ob der Prozessor-Port defekt ist. Um dieses feststellen zu können, sollten Sie jedoch schon einige Erfahrung im Umgang mit ICs haben.

Dazu messen Sie die drei Leitungen -LORAM, -HIRAM und -CHAREN an den Pins 27, 28 und 29 des Prozessors. Diese Leitungen sollten nach dem Einschalten auf High liegen. Stellt sich nach dem Einschalten aber ein Low-Pegel an einem dieser Pins ein, so kann der Rechner das Betriebssystem-ROM nicht ordnungsgemäß adressieren. Es wird ausgeblendet und das darunterliegende RAM kommt zum Vorschein. Folglich kann der Computer die RESET-Routine nicht anspringen, was unweigerlich zum "Absturz" führt.

Stellt sich also ein Low-Pegel ein, so können Sie jetzt den Prozessor-Port untersuchen. Dafür löten Sie den Prozessor aus und löten eine 40-beinige Fassung an diese Stelle. Nun müssen Sie die Pins 27, 28 und 29 des Prozessors rechtwinklig abbiegen und ihn zurück in die Fassung stecken, so daß diese Pins keinen Kontakt zur Platine haben. Auf diese Art wird dem Adreß-Manager U17 vorgegaukelt, daß alles in Ordnung ist, da ein offener Eingang an TTL-ICs als High gewertet wird. Wenn der Rechner nach dieser Prozedur arbeitet, so ist ein neuer Prozessor nötig. Schlägt diese Methode fehl, so ist es unumgänglich, den Com-

puter in eine Werkstatt zu geben, da der Fehler kaum mit normalen Mitteln zu lokalisieren ist.

Farbige Zeichen auf dem Bildschirm

Wenn Sie zum Beispiel beim Listen einer Directory oder eines BASIC-Programms ungewollt lauter farbige Zeichen auf dem Bildschirm erhalten, so muß dies nicht unbedingt am VIC liegen. Der Fehler kann auch an dem Netzteil liegen, welches zu wenig Spannung liefert. Um dieses zu überprüfen, messen Sie die Spannung nach oder tauschen Ihr Netzteil aus.

Die Tastatur funktioniert nicht richtig!

In diesem Fall liegt bestimmt ein Fehler der CIA 1 vor, die für die Tastaturabfrage zuständig ist. Gewißheit darüber können Sie sich jedoch mit dem Testprogramm verschaffen.

Der Joystick funktioniert nicht!

Hier liegt der Fehler, sofern keine Störungen der Tastatur vorliegen, ausschließlich am Joystick selbst. Mit Hilfe des Testprogramms können Sie die Joystick-Funktionen überprüfen.

Wenn er nicht richtig lädt!

Hier muß man zwischen Datasette und Floppy unterscheiden. Beim Bandbetrieb gibt es drei Fehlermöglichkeiten:

In der Datasette selbst

Der Tonkopf ist dejustiert. Mit einem kleinen Schraubenzieher kann man versuchen, die Stellung des Tonkopfes anhand der Justageschraube zu justieren (merken Sie sich die ursprüngliche Einstellung, damit Sie, falls dies nicht die Fehlerquelle war, sich nicht die Datasette selbst dejustieren). Eine weitere Möglichkeit ist der verschmutzte Tonkopf, den man mit einer Reinigungskassette oder einem Tonkopfspray säubern kann.

Im Computer

Auch hier muß zwischen zwei Fällen unterschieden werden:

1. Es wurde nicht richtig "gesavet", was auf einen defekten Prozessor-Port zurückzuführen ist. In diesem Fall müssen Sie den Computer in die Reparatur schicken.
2. Es wird nicht richtig geladen, was auf eine defekte CIA 1 schließen läßt. In diesem Fall können Sie, die beiden CIAs vertauschen.

Die Floppy lädt nicht!

Falls der Fehler im Computer liegt, so kann dies nur bei der CIA 2 der Fall sein, da diese für die Kommunikation mit der Floppy zuständig ist. Hier ist es ratsam, die CIA 2 mit der eines Freundes auszutauschen, um sich Gewißheit zu verschaffen. Wenn dies nicht hilft, so müßte der Fehler in der Floppy liegen, die man, wenn es nicht auf eine defekte Sicherung in der Floppy zurückzuführen ist, zu einer Reparaturwerkstatt bringen muß.

Fehler, die nach längerem Betrieb auftreten

Zeitweise oder spontan auftretende Fehler sind in einer Werkstatt immer die unbeliebtesten Defekte. Da kann es dann ohne weiteres vorkommen, daß ein Gerät mehrere Tage im Test ordnungsgemäß funktioniert, obwohl der Kunde als Fehler angegeben hat, daß sich das Gerät nach 1/2 Stunde "verabschiedet". Häufige Ursache dieser Fehler ist ein thermischer Defekt in einem Bauteil. Sollte Ihr C64 irgendwann einmal solche Symptome zeigen, so empfiehlt sich die Anschaffung einer großen Dose Kältespray. Dieses Spray ist im Elektronikfachhandel für ein paar DM zu erhalten. Durch einfaches Ansprühen lassen sich die Bauteile auf bis zu -40 Grad abkühlen. Dabei hat sich das folgende Prinzip bewährt:

Nachdem die Tastatur entfernt ist, wird die Rechnerplatine mit einem normalen Haarfön gleichmäßig erhitzt. Sobald der Rechner einen Fehler zeigt, werden die einzelnen Bauteile systematisch mit dem Spray gekühlt und der Rechner nach jedem Bau-

teil aus- und wieder eingeschaltet. Sobald nach dem Einschalten der Rechner wieder funktioniert, werden die zuletzt abgekühlten Bauteile noch einmal erwärmt, um zu sehen, ob wirklich eines dieser Bauteile den Defekt verursachte. Auf diese Weise kann man nun das defekte Bauteil immer weiter einkreisen und zuletzt austauschen.

Leider sind nicht nur die Bauteile als Fehlerursache möglich. Auch die Leiterplatte kann als Ursache in Frage kommen. Durch die Erwärmung dehnt sie das Material aus, und obwohl diese Ausdehnung sehr gering ist, können winzige Haarrisse entstehen, die dann zum Versagen des Gerätes führen. Diese Haarrise zu finden ist nur mit viel Glück möglich. Glücklicherweise ist das im C64 verwendete Leiterplattenmaterial von sehr guter Qualität, so daß diese Fehler ausgesprochen selten sind.

Eine Ursache für sporadisch auftretende Fehler können auch die RAMs sein. Mit dem in diesem Kapitel abgedruckten Testprogramm ist es möglich, das gesamte RAM des C64 zu prüfen und das vermutlich fehlerhafte RAM-IC anzuzeigen.

Das Herausnehmen von ICs

Diese Anleitung gilt nur für ICs, die gesockelt sind. Sollte dies nicht der Fall sein, überlassen Sie diese Arbeit einem Fachmann.

Nehmen Sie einen kleinen Schraubenzieher, und schieben Sie ihn vorsichtig unter die eine der kurzen Seiten des ICs. Drücken Sie den Schraubenzieher nun allmählich nach unten, um das IC an der Stelle etwas aus der Fassung zu heben. Genauso verfahren Sie an der anderen Seite. Achten Sie dabei darauf, daß Sie das IC nicht einseitig zu weit anheben, weil dies ein Verbiegen der noch steckenden Beinchen auf der anderen Seite des ICs zur Folge haben könnte. Wenn Sie das IC nun so weit aus der Fassung gehoben haben, daß Sie es mühelos mit Daumen und Zeigefinger, die die beiden Enden des ICs halten, herausziehen können, dann nehmen Sie es an beiden Enden gleichmäßig ziehend aus der Fassung heraus.

Fassen Sie die Beinchen des ICs nicht an, weil Sie statisch aufgeladen sein können, was eine Zerstörung des ICs zur Folge haben könnte.

Wie stelle ich mir meine Tastatur strammer?

Dieser kleine Trick ist sehr nützlich, wenn man die Tastatur satt hat. Alle Tasten des C64 sind mit Federn ausgestattet, die die Tasten vom Kontakt auf der Tastaturplatine wegdrücken. Die Tasten sind auf einem kleinen Stäbchen festgesteckt. Mit Hilfe des Schraubenziehers kann man die Tasten vom Keyboard lösen.

Man schiebt den Schraubenzieher unter eine Taste und drückt dann vorsichtig den Schraubenzieher nach unten, um die Taste aus der Halterung zu hebeln. Gleichzeitig zieht man die Taste mit Daumen und Zeigefinger nach oben. Nachdem Sie die Taste entfernt haben, nehmen Sie die Feder heraus, die ca. 1 cm lang ist. Sie können sie jetzt vorsichtig auseinanderziehen. Es ist nicht ratsam, sie länger als 1.5 cm zu ziehen, da der Anschlag sonst zu hart wird.

Nachdem Sie die Feder präpariert haben, setzen Sie sie wieder auf ihren alten Platz. Dann stecken Sie die Taste auf den Pin und pressen sie nach unten, bis sie einrastet.

Die Taste hat nun einen härteren Anschlag. Am besten stellt man sich die Tasten härter, die am meisten benutzt werden, zum Beispiel die <Return>-, <Run/Stop>- und <Restore>-Tasten.

Wie baue ich einen RESET-Taster ein?

Bevor wir in die Praxis übergehen, wollen wir Ihnen zuvor erst die Arbeitsweise eines RESET erklären. Beim Auslösen eines Hardware-RESET geschieht das gleiche wie beim Einschalten des Computers. Die RESET-Leitung des Computers, die mit den wichtigsten Bauteilen verbunden ist, wird kurzzeitig auf Low (MASSE) gelegt. Daraufhin springt der Prozessor die RESET-Routine, deren Vektor in Speicheradresse $FFFC und $FFFD liegt, an. In dieser Routine werden die wichtigsten Bausteine initialisiert, während der Speicherinhalt zum größten Teil erhalten

bleibt. In unserer Anleitung haben wir die einfachste Möglichkeit zum Bau eines RESET-Tasters gewählt. Es werden die Leitungen RESET und MASSE des User-Ports per Taster miteinander verbunden. Zum Bau benötigen Sie folgende Bauteile: Einen Taster und zwei dünne Kabel.

Falls Sie nichts an Ihren Computer anlöten wollen, dann besorgen Sie sich noch einen User-Port-Stecker und löten die beiden Kabel nicht direkt an den User-Port, sondern an den Stecker. Jetzt löten Sie die beiden Kabelenden an Pin 1 und 3 des User-Ports oder des Steckers. Drücken Sie nun auf den Taster, und Sie werden feststellen, daß Ihr Rechner einen RESET ausführt. Den gleichen Vorgang können Sie auch mit SYS64738 vom BASIC aus erzielen.

Das Testprogramm

Ans Ende dieses Kapitels haben wir noch ein kleines Testprogramm gehängt, mit dessen Hilfe Sie das RAM, den Soundchip und Ihren Joystick überprüfen können. Außerdem enthält es noch ein Testbild, mit dem Sie die Farben und den Kontrast Ihres Fernsehers oder Monitors einstellen können. Das Progamm ist in Assembler geschrieben und liegt hier als BASIC-Lader vor.

```
5 N=49152
10 READX:IFX=-1THEN30
20 S=S+X:POKEN,X:N=N+1:GOTO 10
30 IF S<>124998 OR N<>50359 THEN PRINT"FEHLER IN DATAS":END
40 SYS49152
101 DATA 169,0,141,32,208,141,33,208,170,169,5,141,134,2,189,58,192,32,2
10
102 DATA 255,232,201,0,208,245,32,62,241,240,251,201,49,208,3,76,154,192
,201
103 DATA 50,208,3,76,125,195,201,51,208,3,76,59,194,201,52,208,201,76,56
,193
104 DATA 147,13,32,32,32,32,32,32,32,67,72,69,67,75,80,82,79,71,82,65,77
,77
105 DATA 32,13,13,13,13,32,32,18,32,49,32,146,32,84,69,83,84,66,73,76,68
,13
106 DATA 13,32,32,18,32,50,32,146,32,83,79,85,78,68,13,13,32,32,18,32,51
,32
107 DATA 146,32,82,65,77,84,69,83,84,13,13,32,32,18,32,52,32,146,32,74,7
9
108 DATA 89,83,84,73,67,75,13,13,0,169,147,32,210,255,160,28,162,0,189,2
23
```

```
109 DATA 192,32,210,255,232,201,0,208,245,136,208,240,162,0,160,40,189,2
55
110 DATA 192,32,210,255,136,208,250,232,224,20,208,240,162,0,189,21,193,
240
111 DATA 7,32,210,255,232,76,197,192,162,18,160,12,24,32,10,229,32,86,19
5
112 DATA 76,0,192,18,154,32,32,5,32,32,28,32,32,159,32,32,156,32,32,30,3
2
113 DATA 32,31,32,32,158,32,32,152,32,32,153,32,32,0,18,154,32,5,32,28,3
2
114 DATA 159,32,156,32,30,32,31,32,158,32,152,32,153,32,0,19,5,18,29,29,
29
115 DATA 29,29,29,29,29,29,17,84,69,83,84,66,73,76,68,32,68,69,83,32,68,
65
116 DATA 84,65,83,65,84,13,0,162,0,189,211,193,32,210,255,232,201,0,208,
245
117 DATA 162,13,160,16,24,32,240,255,162,0,189,44,194,32,210,255,232,201
,0
118 DATA 208,245,162,13,160,16,24,32,240,255,162,0,173,0,220,41,1,208,13
,189
119 DATA 9,194,32,210,255,232,201,0,208,245,240,205,173,0,220,41,2,208,1
3
120 DATA 189,16,194,32,210,255,232,201,0,208,245,240,185,173,0,220,41,4,
208
121 DATA 13,189,23,194,32,210,255,232,201,0,208,245,240,165,173,0,220,41
,8
122 DATA 208,13,189,30,194,32,210,255,232,201,0,208,245,240,25,173,0,220
,41
123 DATA 16,208,13,189,37,194,32,210,255,232,201,0,208,245,240,5,32,62,2
41
124 DATA 208,3,76,69,193,76,0,192,147,13,83,84,69,67,75,69,78,32,83,73,6
9
125 DATA 32,68,69,78,32,74,79,89,83,84,73,67,75,32,73,78,32,80,79,82,84,
32
126 DATA 35,50,13,17,18,69,78,68,69,146,32,61,32,84,65,83,84,69,0,79,66,
69
127 DATA 78,32,32,0,85,78,84,69,78,32,0,76,73,78,75,83,32,0,82,69,67,72,
84
128 DATA 83,0,70,69,85,69,82,32,0,32,32,32,32,32,32,32,157,157,157,157,1
57
129 DATA 157,157,0,120,162,0,189,7,195,32,210,255,232,201,0,208,245,162,
0
130 DATA 181,0,141,144,4,160,0,200,208,253,213,0,240,3,76,235,194,232,20
8
131 DATA 236,162,0,189,53,195,32,210,255,232,201,0,208,245,162,0,189,64,
195
132 DATA 32,210,255,232,201,0,208,245,162,0,189,0,1,160,0,200,208,253,14
1
133 DATA 144,4,157,0,1,221,0,1,208,93,232,208,234,162,0,189,53,195,32,21
0
134 DATA 255,232,201,0,208,245,162,0,189,75,195,32,210,255,232,201,0,208
,245
```

```
135 DATA 120,169,48,133,1,162,0,160,4,134,251,132,252,162,0,160,251,161,
251
136 DATA 129,251,141,144,4,193,251,208,36,232,208,242,230,252,136,208,23
7
137 DATA 169,55,133,1,162,0,189,53,195,32,210,255,232,201,0,208,245,169,
13
138 DATA 32,210,255,32,86,195,76,0,192,169,55,133,1,162,0,189,40,195,32,
210
139 DATA 255,232,201,0,208,245,169,13,32,210,255,32,86,195,76,0,192,147,
13
140 DATA 32,32,32,32,32,32,32,32,82,65,77,84,69,83,84,17,17,17,17,17,13,
13
141 DATA 90,69,82,79,80,65,71,69,0,32,32,32,32,32,32,70,69,72,76,69,82,0
,32
142 DATA 32,32,32,32,32,32,32,79,75,0,13,13,83,84,65,80,69,76,32,32,0,13
,13
143 DATA 54,52,75,32,82,65,77,32,0,162,0,189,105,195,32,210,255,232,201,
0
144 DATA 208,245,32,62,241,240,251,96,18,17,17,17,84,65,83,84,69,32,68,8
2
145 DATA 85,69,67,75,69,78,146,0,32,68,229,162,0,134,124,169,8,133,125,3
2
146 DATA 49,196,166,124,169,15,141,24,212,169,1,157,0,212,133,126,169,22
,157
147 DATA 1,212,169,34,157,5,212,169,133,157,6,212,169,15,157,3,212,157,2
,212
148 DATA 6,125,32,2,196,165,125,9,1,166,124,157,4,212,32,232,195,165,125
,157
149 DATA 4,212,32,232,195,32,243,195,176,231,36,125,48,3,76,176,195,169,
0
150 DATA 157,4,212,165,124,24,105,7,133,124,201,21,144,159,76,0,192,160,
0
151 DATA 162,5,136,208,253,202,208,250,96,166,124,230,126,165,126,240,5,
157
152 DATA 1,212,56,96,24,96,32,87,196,169,32,36,125,112,14,48,17,208,5,16
2
153 DATA 0,76,32,196,162,14,76,32,196,162,30,76,32,196,162,44,189,105,19
6
154 DATA 240,7,32,210,255,232,76,32,196,169,13,76,210,255,32,68,229,32,9
7
155 DATA 196,165,124,240,10,201,7,240,3,162,59,44,162,56,44,162,53,32,32
,196
156 DATA 162,62,32,32,196,169,13,32,210,255,76,210,255,56,32,240,255,160
,7
157 DATA 24,76,240,255,162,5,160,7,24,76,240,255,68,82,69,73,69,67,75,45
,87
158 DATA 69,76,76,69,0,83,69,65,71,69,90,65,72,78,45,87,69,76,76,69,0,82
,69
159 DATA 67,72,69,67,75,45,87,69,76,76,69,0,82,65,85,83,67,72,69,78,0,49
,46
160 DATA 0,50,46,0,51,46,0,32,32,84,79,78,71,69,78,69,82,65,84,79,82,0,9
,-1
```

10. Die Datasette VC 1530 - Tips und Tricks

Die Datasette VC 1530 ist ein Gerät zur Speicherung von Daten auf Magnetband. Sie funktioniert wie ein Kassettenrekorder. Im folgenden werden Tips zu deren Handhabung aufgeführt.

10.1 Die Befehle zur Datasetten-Handhabung

Das Commodore-BASIC stellt Ihnen acht Befehle zum Arbeiten mit der Datasette zur Verfügung:

1. SAVE Programme speichern
2. LOAD Programme laden
3. VERIFY Programme auf Richtigkeit überprüfen
4. OPEN Eine Datei eröffnen
5. CLOSE Eine Datei schließen
6. PRINT# Einen Ausdruck senden
7. INPUT# Einen Ausdruck einlesen
8. GET# Ein einzelnes Byte einlesen

Im folgenden möchte ich die Befehle näher beschreiben.

10.1.1 SAVE

Mit diesem Befehl veranlassen Sie den Computer, das im Speicher befindliche Programm abzuspeichern. Um zu wissen, an welcher Stelle das Programm im Speicher steht, gibt es in Ihrem Rechner einige Speicherstellen, in denen steht, wo das Programm beginnt und wo es endet.

Der Rechner legt sich nämlich sofort nach dem Einschalten ein "Merkheft" an. Dieses Heft hat vier Seiten (0 - 3), sogenannte "Pages", und liegt am Anfang des ganzen Speicherbereichs. Jeweils 256 Speicherstellen werden aufgrund der prozessoreigenen Speicherverwaltung zu einer Page zusammengefaßt, da jede Speicherstelle oder auch jedes Byte genau 256 (0 - 255) ver-

schiedene Werte annehmen kann. In der nullten Seite, also von 0 bis 255, merkt sich der Rechner die Werte, die er sehr oft braucht.

In den Stellen 43, 44 steht immer der aktuelle Start des BASIC-Speichers und in 45, 46 das Ende eines BASIC-Programms. Schauen Sie doch einmal nach. Geben Sie PRINT PEEK (43), PEEK (44), PEEK (45), PEEK (46) ein. Auf dem Bildschirm erscheint dann

```
1  8  3  8
```

wenn Sie kein Programm geladen haben. Da der Rechner 65.535 Speicherstellen hat, werden Adressen immer durch 2 Bytes ausgedrückt. Das erste Byte, das LSB (Last Significant Byte), gibt die Position innerhalb einer Seite an, das zweite, das MSB (Most Significant Byte), die Seitennummer. Unser BASIC-Bereich fängt also auf Seite 8 an der ersten Speicherstelle an, also mit der Speicherstelle 8*256+1 = 2.049.

"Wieso stehen aber in den Speicherstellen 45, 46 die Werte 3 und 8, ich habe doch gar kein Programm im Speicher ?", werden Sie jetzt fragen. Das hat folgenden Grund: Damit der Rechner weiß, wo ein BASIC-Programm zu Ende ist, hängt er immer 2 Nullen an das Ende, um es zu markieren. Sie haben also jetzt ein Programm im Speicher, welches sozusagen nur aus einer Programmendemarkierung besteht. Sie können dies leicht überprüfen. Wenn Sie

```
PRINT PEEK (PEEK(45)+PEEK(46)*256 -1)
```

eingeben, erhalten Sie 0. Geben Sie jetzt einmal eine BASIC-Zeile ein, z.B.

```
10 PRINT "HALLO"
```

und schließen Sie die Eingabe mit <Return> ab. Wenn Sie jetzt wie oben die Inhalte der Speicherstellen 43, 44, 45, 46 auslesen, erhalten Sie:

```
1816  8
```

Der Rechner hat Ihrem Programm entsprechend die Eintragungen in seinem Merkheft geändert. Nach dieser Exkursion in die Speicherverwaltung kehren wir zum SAVE-Befehl zurück.

Durch diesen Befehl veranlassen wir also den Rechner, alles, was zwischen den eben genannten Zeigern steht, auf Band abzuspeichern. Dem Computer ist es dabei egal, ob auf dem Band schon etwas abgespeichert ist oder nicht. Wie bei einem normalen Kassettenrekorder wird die Kassette einfach überschrieben. Damit Sie aber das abgespeicherte Programm wieder finden können, haben Sie die Möglichkeit, dem abzuspeichernden Programm einen Namen zu geben. Die Syntax ist dann folgende:

```
SAVE"NAME"
```

Sie haben natürlich auch die Möglichkeit, den String "NAME" vorher in eine String-Variable einzulesen und ein Programm mit folgender Syntax abzuspeichern:

```
A$="NAME"
SAVEA$
```

Diese Möglichkeit kann wichtig sein, wenn Sie diesen Befehl innerhalb von Programmen nutzen. Aber darüber finden Sie im Unterkapitel 10.4 mehr. Nach dem Namen haben Sie dann noch die Möglichkeit, eine Geräteadresse und eine Sekundäradresse, jeweils durch Komma getrennt, folgen zu lassen. Die genaue Bedeutung der Sekundäradresse finden Sie in Unterkapitel 10.2 beschrieben. Die Geräteadresse sagt dem Computer, welches Peripheriegerät er wählen soll. Sie stellt sozusagen die Hausnummer der verschiedenen Geräte dar. Die Hausnummer der Datasette ist 1, die einer Diskettenstation 8 oder 9.

Damit Sie aber nicht soviel tippen müssen, ist Ihr Computer so programmiert, daß er immer die Datasette auswählt, wenn keine Angabe über die Geräteadresse gemacht wird.

10.1.2 LOAD

Mit diesem Befehl können Sie das einmal Abgespeicherte wieder laden. Geben Sie dem Rechner keinen Namen an, lädt er das nächste Programm, welches er finden kann. Sie können bei diesem Befehl genauso wie bei SAVE einen Namen angeben. Auch können Sie hier wieder eine Geräte- und Sekundäradresse folgen lassen. Hierfür gilt das gleiche wie beim SAVE-Befehl.

Nach jedem LOAD-Befehl im Direktmodus werden vom Rechner automatisch die Eintragungen in seinem Merkheft entsprechend dem geladenen Programm geändert. Dies ist zu beachten, wenn Sie Maschinenprogramme laden. Wenn Sie z.B. ein Maschinenprogramm laden, das am Ende des BASIC-Speichers liegt, werden dementsprechend auch die BASIC-Endvektoren auf das BASIC-RAM-Ende gesetzt. Eine weitere Eingabe wird danach mit "?OUT OF MEMORY ERROR" vom Rechner kommentiert. Um nun weitere Eingaben machen zu können, müssen Sie entweder in die Zeiger 45,46 den alten Wert poken, oder Sie geben einen NEW-Befehl, damit das im BASIC-Speicher befindliche Programm, nicht aber das Maschinenprogramm gelöscht wird.

Sie können den NEW-Befehl umgehen, wenn Sie das Maschinenprogramm im Programm-Modus laden. Näheres lesen Sie dazu im Unterkapitel 10.4.

10.1.3 VERIFY

Dieser Befehl arbeitet ähnlich wie der LOAD-Befehl. Der einzige Unterschied ist, daß durch diesen Befehl das Programm nicht in den Speicher geschrieben wird, sondern Byte für Byte mit dem im Speicher stehenden verglichen wird. Dadurch können Sie testen, ob eine Programmspeicherung erfolgreich war.

War die Speicherung erfolgreich und hat auch die Kassette keinen Fehler, erscheint OK auf dem Bildschirm. Wird bei dem Vergleich ein Fehler festgestellt, erhalten Sie die Meldung "?VERIFY ERROR".

10.1.4 OPEN

Bis jetzt habe ich nur über das Laden und Speichern von Programmen geschrieben. Sie haben aber auch die Möglichkeit, Daten zu speichern und zu laden. Um dem Rechner mitzuteilen, daß Sie Daten laden oder speichern wollen, müssen Sie mit dem OPEN-Befehl eine Datei eröffnen.

Mit diesem Befehl teilen Sie dem Computer alle Parameter mit, die er benötigt, nämlich die Gerätenummer, den File-Namen und die Sekundäradresse, die darüber etwas aussagt, ob geladen oder gespeichert werden soll. Um eine Datei zu eröffnen, müssen Sie folgenden Befehl eingeben:

```
OPEN lf,GA,SA,"NAME"
```

Mit "NAME" können Sie der Datei wie bei SAVE und LOAD einen Namen geben.

SA steht für Sekundäradresse: Über diese Zahl teilen Sie dem Computer mit, ob Sie Daten senden (= 1) oder empfangen (= 0) wollen. Geben Sie eine 2 ein, wird nach dem Abspeichern noch ein EOT (End Of Tape) Block geschrieben.

GA steht für Geräteadresse: Diese Zahl sagt dem Rechner, mit welchem Gerät er arbeiten soll. Dabei bedeutet

```
0= Tastatur
1= Kassette
3= Bildschirm
4,5 = Drucker
8,9 = Diskette
```

"lf" steht für logische File-Nummer: Diese Zahl, sie kann zwischen 0 und 255 liegen, stellt den Index dieser OPEN-Anweisung dar. Damit nicht bei jedem Ein- und Ausgabebefehl alle Parameter mit übergeben werden müssen, legt sich der Computer eine Tabelle an, in der die Parameter der OPEN-Anweisung unter dieser Indexnummer abgelegt sind. Nachdem wir nun eine Datei eröffnet haben, kommen wir zu den Befehlen der Datenein- und ausgabe.

10.1.5 PRINT# und CLOSE

Mit dem PRINT#-Befehl ist es möglich, einzelne Daten auf Band zu schreiben. Zur besseren Erklärung geben Sie einmal folgendes Programm ein:

```
10 OPEN 1,1,1,"DATEN-FILE"
20 PRINT "DATEI WURDE EROEFFNET"
30 PRINT:PRINT"GEBEN SIE DATEN EIN"
40 INPUT"DATEN";A$
50 PRINT#1,A$
60 A=A+1
70 PRINT A"DATEN WURDEN ZWISCHENGESPEICHERT"
80 PRINT:PRINT"WEITERE DATEN"
90 GET F$: IF F$ = "" THEN 90
100 IF F$ = "J" THEN 40
110 CLOSE1
120 PRINT:PRINT"DATEI WURDE GESCHLOSSEN"
130 END
```

Legen Sie nun eine leere Kassette in Ihren Rekorder und starten das Programm mit RUN. Es erscheint sofort die Meldung PRESS RECORD & PLAY ON TAPE. Wenn Sie dieser Meldung Folge leisten, wird entsprechend der OPEN-Anweisung in Zeile 10 ein Dateikopf auf Band geschrieben, der den Namen der Datei enthält. Danach hält der Rekorder an, und Sie werden durch Zeile 40 gebeten, Daten einzugeben.

In der Zeile 50 wird dann durch den PRINT#-Befehl der eingegebene String dann zwischengespeichert. Die Nummer nach der PRINT#-Anweisung, es ist die logische File-Nummer, stellt den Bezug zur OPEN-Anweisung her. Daß der String nicht sofort auf Band geschrieben wird, können Sie daran sehen, daß sich das Band noch nicht bewegt.

Erst, wenn Sie eine ganze Reihe von Daten eingegeben haben, setzt sich das Band in Bewegung, und alle bis dahin eingegebenen Daten werden auf das Band übertragen. Auf den Zwischenspeicher, es handelt sich um den Kassettenpuffer, werde ich noch im Unterkapitel 10.5 genauer eingehen. Wenn Sie nun auf die Abfrage in Zeile 100 mit "J" antworten, können Sie weitere Daten eingeben. Ist der Puffer voll, können Sie feststellen, daß eine Pause entsteht und sich das Band kurzzeitig be-

wegt. Jetzt wird der Puffer auf Band geschrieben und daran anschließend mit "Space" (CHR$(32)) beschrieben, um neue Daten aufzunehmen.

Wenn Sie die Eingabe beenden, verzweigt der Rechner zum CLOSE-Befehl in Zeile 110. Die Zahl hinter diesem Befehl ist wieder eine logische File-Nummer und sagt dem Computer, daß er die mit der logischen File-Nummer 1 geöffnete Datei auch schließen soll. Das macht er, indem er alle Eintragungen, die er in seinem Merkheft unter dem Index 1 notiert hat, löscht. Der Rechner schreibt nun hinter die letzte Eintragung ein Null-Byte, um das Datenende zu kennzeichnen, und speichert den Puffer auf Band. Es ist also sehr wichtig, eine Datei wieder zu schließen, da sonst die letzten Daten verlorengehen.

Bis jetzt haben wir nur Strings bzw. den Inhalt von String-Variablen abgespeichert. Die Frage ist nun: "Wie verarbeitet der Rechner numerische Ausdrücke und Variablen?" Solche Ausdrücke werden abgespeichert, als wären es Strings. D.h., der Rechner führt bei jedem PRINT#-Befehl automatisch einen STR$-Befehl durch. Es ist also das gleiche, ob Sie

```
PRINT#1,55  oder
A$=STR$(55):PRINT#1,A$
```

schreiben. Aus diesem Grund können Sie später jeden numerisch abgespeicherten Wert auch in eine String-Variable einlesen. Eine weitere, wichtige Sache ist das Format einer PRINT#-Anweisung, wenn Sie mit einer Anweisung mehrere Ausdrücke abspeichern wollen. Auf dieses Problem möchte ich aber zum besseren Verständnis erst nach der Beschreibung INPUT#-Befehls kommen.

10.1.6 INPUT#

Dieser Befehl verhält sich vollkommen analog zum "normalen" INPUT-Befehl, nur daß hier nicht die Tastatur als festes Empfangsgerät voreingestellt ist. Wenn Sie z.B. mit

```
OPEN1,0
```

(0=Tastatur) die Tastatur als Sender definieren, verhält sich INPUT#1,A$ genauso wie INPUTA$, mit dem kleinen Unterschied, daß kein Fragezeichen ausgedruckt wird. Wie sich nun INPUT# in Hinsicht auf den Kassettenrekorder verhält, sehen wir uns am besten wieder mit einem kleinen Beispielprogramm an. Spulen Sie Ihre Kassette zu dem Punkt zurück, wo Sie die Datei DATEN-FILE abgespeichert haben, und geben Sie folgendes Programm ein:

```
10 OPEN 1,1,0,"DATEN-FILE"
20 PRINT "DATEI WURDE EROEFFNET"
30 PRINT:PRINT"DATEN WERDEN EINGELESEN"
40 INPUT#1,A$
50 PRINTA$
60 A=A+1
70 PRINT A"DATEN WURDEN EINGELESEN"
80 PRINT:PRINT"WEITERE DATEN"
90 GET F$: IF F$ = "" THEN 90
100 IF F$ = "J" THEN 40
110 CLOSE1
120 PRINT:PRINT"DATEI WURDE GESCHLOSSEN"
130 END
```

Wenn Sie nun das Programm starten und laut Anweisung die <Play>-Taste drücken, sehen Sie, daß das Band einige Zeit läuft und dann anhält. Ihr Rechner hat nun aufgrund des OPEN-Befehls die Datei "DATEN-FILE" gesucht und dann aufgrund des INPUT#-Befehls in Zeile 40 den ersten Datenblock eingelesen und den ersten String der Variablen A$ übergeben, die über Zeile 50 auf dem Bildschirm angezeigt wird.

Die einzelnen Daten stehen jetzt genauso im Puffer, wie Sie sie beim Abspeichern hereingeschrieben haben. Hinter jedem String steht aufgrund der PRINT#-Anweisung ein "CARRIAGE RETURN" (CHR$(13)), also ein Zeilenvorschub. Der PRINT#-Befehl schreibt die Daten genauso in den Puffer, wie ein PRINT-Befehl sie auf den Bildschirm schreiben würde. Der INPUT#-Befehl erkennt genauso wie der INPUT-Befehl durch ein Komma oder <Return>, daß hier der Ausdruck zu Ende ist.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	...27	28	29	30	31
Inhalt	S	T	R	1	CR	S	T	R	2	CR	S	T	...R	5	CR	00	

Beantworten Sie jetzt die Frage, ob weitere Daten eingelesen werden sollen. Bei unserem Beispielprogramm geben Sie "J" ein. Jetzt können Sie String für String wieder einlesen und auf dem Bildschirm ausdrucken lassen, bis Sie die Meldung erhalten "?STRING TOO LONG ERROR".

Diese Meldung erhalten Sie, weil Sie inzwischen den letzten, abgespeicherten String eingelesen haben und nun versuchen, einen weiteren einzulesen. Der Computer sucht das Ende des Strings, also ein Komma oder "CARRIAGE RETURN" (kurz CR), findet aber in den nächsten folgenden 80 Speicherstellen keins. (Sie können maximal 80 Zeichen mit einer INPUT- bzw. INPUT#-Anweisung einlesen.) Daraus schließt der Computer, daß hier ein String länger als 80 Zeichen vorliegt und gibt die Fehlermeldung aus.

Damit Sie aber nicht bei einem "normalen" Dateiprogramm immer diese Fehlermeldung bekommen, gibt es eine Möglichkeit, das Dateiende mit Hilfe der Statusvariablen zu erkennen. Wie Sie das programmieren müssen, finden Sie im Unterkapitel 10.3. Diese Fehlermeldung erhalten Sie aber nicht nur, wenn Sie über das Dateiende hinaus zu lesen versuchen, sondern auch dann, wenn die Daten in einem falschen Format abgespeichert wurden.

Bei der Besprechung des PRINT#-Befehls haben wir immer nur einen Ausdruck pro PRINT#-Anweisung abgespeichert, was zur Folge hatte, daß automatisch immer ein CR angehängt wurde. Es ist aber auch möglich, mehrere Ausdrücke mit einer PRINT#-Anweisung abzuspeichern. Dabei muß aber darauf geachtet werden, daß die einzelnen Ausdrücke voneinander getrennt werden. Schreiben Sie z.B.

```
PRINT#1,A$,B$
```

mit A$="STRING1" und B$="STRING2", werden die beiden Ausdrücke genauso in den Puffer geschrieben, wie sie mit einer PRINT-Anweisung auf den Bildschirm geschrieben würden, nämlich

```
Byte    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Inhalt  S T R I N G 1 S T R I  N  G  2  CR
```

Wenn Sie aber folgende Anweisungen geben:

```
T$=","
PRINT#1,A$;T$;B$
```

erhalten Sie

```
Byte    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Inhalt  S T R I N G 1 , S T R  I  N  G  2  CR
```

Jetzt sind die Strings eindeutig voneinander getrennt und werden von einer INPUT#1,A$,B$-Anweisung richtig eingelesen. Auf eine beachtenswerte Besonderheit muß ich aber noch hinweisen. Das Komma reicht zwar aus, um die Daten für eine eindeutige Variablenzuweisung mit INPUT# zu gewährleisten, aber es reicht nicht aus, dem Computer zu sagen, daß hier eine Eintragung zu Ende ist und daß ab dem nächsten Byte eine neue beginnt. Das klingt zwar sehr ähnlich, ist es aber nicht. Deshalb möchte ich im folgenden etwas näher darauf eingehen.

Man kann den Computer mit einer großen Firma mit vielen Abteilungen vergleichen. Das Verarbeiten von Daten beim Lesen und Schreiben überläßt er zwei Abteilungen.

1. Abteilung - Variablenverarbeitung

Diese Abteilung sorgt dafür, daß ein Ausdruck, der irgendwo im Speicher steht (Kassettenpuffer, Bildschirmspeicher, Programm etc.), der richtigen Variablen im richtigen Format zugeordnet wird.

2. Abteilung - Speicherplatzverwaltung

Diese Abteilung sagt der ersten Abteilung immer, ab welchem Speicherplatz der für sie interessante Ausdruck steht. Jede der beiden Abteilung hat nun ihre eigene Art, das Ende eines Ausdrucks und somit den Beginn eines neuen zu suchen. Der ersten Abteilung ist es egal, ob sie ein Komma oder einen CR findet.

Bei beiden weiß sie, daß hier ein Ausdruck zu Ende ist und danach ein neuer beginnt. Die zweite Abteilung kümmert sich um das Komma überhaupt nicht. Sie erkennt nur den CR als Trennungszeichen an. D.h., sie gibt als Startpunkt für einen neuen Ausdruck immer die Stelle nach einem CR an.

Am besten veranschaulicht man sich das mit einem kleinen Beispielprogramm. Geben Sie einmal die folgenden Zeilen ein, legen eine Leerkassette in Ihren Rekorder und starten das Programm mit RUN.

```
10 OPEN1,1,1,"TEST"
20 T$=","
30 A1$="STRING1" : A2$="STRING2"
40 A3$="STRING3" : A4$="STRING4"
50 PRINT#1,A1$;T$;A2$;T$;A3$
60 PRINT#1,A4$
70 CLOSE1
80 END
100 OPEN1,1,0,"TEST"
110 INPUT#1,A$
120 PRINT A$
130 INPUT#1,A$
140 PRINT A$
150 CLOSE 1
160 END
```

Nachdem die Datei TEST abgespeichert wurde, spulen Sie Ihre Kassette an den Start der Datei zurück und starten das Programm mit RUN100. Wenn der Rechner die Datei eingelesen hat, stehen die Strings folgendermaßen im Speicher:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
S T R I N G 1 , S T R I N G 2 , S T R I

20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
N G 3 CR S T R I N G 4 CR 00
```

Auf dem Bildschirm erhalten Sie:

```
STRING1
STRING4
```

Abteilung zwei hat beim ersten INPUT# seinen Zeiger auf den ersten String gesetzt, und Abteilung eins hat ihn dann A$ zuge-

ordnet. Da keine weiteren Ausdrücke bei dieser INPUT#-Anweisung eingelesen werden sollen, wird die Ausführung wieder an Abteilung zwei übergeben. Bei der nächsten INPUT#-Anweisung sucht Abteilung zwei das nächste CR und findet es vor STRING4. Diesen Startpunkt übergibt sie der Abteilung eins, welche den Ausdruck "STRING4" wiederum A$ zuweist. Ändern Sie jetzt einmal Zeile 110 und 120, indem Sie ",B$,C$" anhängen. Wenn Sie nun die Datei erneut einlesen, erhalten Sie auf dem Bildschirm:

```
STRING1      STRING2      STRING3
STRING4
```

Jetzt weist Abteilung eins in Zeile 110 erst die beiden folgenden Ausdrücke den Variablen B$ und C$ zu, ehe sie das Kommando an Abteilung zwei zurückgibt.

Kurz:

Der Zeiger der Abteilung eins wird durch jede Variablenzuweisung auf das nächste Trennungszeichen gesetzt (Komma oder CR). Der Zeiger der Abteilung zwei wird durch jede INPUT#-Anweisung auf das nächste Trennungszeichen gesetzt (CR).

Zusammenfassend kann man also sagen, daß ein Komma so lange als Trennung ausreicht, wie Sie die Daten im gleichen Format einlesen, wie Sie sie abgespeichert haben. Wollen Sie aber beim Einlesen frei sein vom Eingabeformat, dann definieren Sie den Trennungs-String mit CR.

```
20 T$=CHR$(13)
```

Bei so abgespeicherten Daten erhalten Sie immer das gleiche Ergebnis, egal, ob Sie mit einer INPUT#-Anweisung eine oder mehrere Variablen einlesen. Bis jetzt habe ich nur über die Trennung von einzelnen Ausdrücken geschrieben. Bei manchen Anwendungen kann es aber sinnvoll sein, einzelne Ausdrücke nicht zu trennen. Dies erreicht man wie bei dem PRINT-Befehl auf dem Bildschirm dadurch, daß man eine PRINT#-Anweisung mit einem Semikolon abschließt. Probieren Sie es einmal mit

dem kleinen Testprogramm von oben. Ändern Sie die Zeilen folgendermaßen ab:

```
10 OPEN1,1,1,"TEST2"
50 PRINT#1,A1$;A2$;A3$;
100 OPEN1,1,0,"TEST2"
```

Löschen Sie Zeile 130 und 140. Wenn Sie mit RUN die Datei TEST2 abspeichern und sie mit RUN100 einladen, erhalten Sie:

```
STRING1STRING2STRING3STRING4
```

Ergeben sich aber durch die Aneinanderreihung Strings, die länger als 80 Zeichen sind, können Sie sie nicht mehr mit der INPUT#-Anweisung einlesen. Solche Strings können Sie dann nur noch Byte für Byte mit der GET#-Anweisung einlesen.

10.1.7 GET#

Auch dieser Befehl arbeitet vollkommen analog dem GET-Befehl. Mit ihm können Sie byteweise Daten einlesen. Geben Sie einmal folgendes Programm ein:

```
10 OPEN1,1,0,"TEST"
20 GET#1,A$
30 PRINTA$;
40 GETA$: IF A$="" THEN 40
50 GOTO20
```

Spulen Sie nun zum Beginn der Datei TEST zurück, und starten Sie das Programm. Nachdem der Computer die Datei gefunden hat, erscheint buchstabenweise folgendes Bild:

```
STRING1      STRING2      STRING3
STRING4
```

Wenn nun noch weitere Zeichen eingelesen werden, können Sie keine Veränderung auf dem Bildschirm erkennen, da der Puffer mit "SPACE" aufgefüllt ist. Durch folgendes Programm können Sie die Zeichen sichtbar machen:

```
30 PRINT A$,ASC(A$+CHR$(0))
```

Dadurch erreichen Sie, daß jedes Zeichen als Buchstabe und ASCII-Wert ausgedruckt wird. Trifft die GET#-Anweisung auf ein Null-Byte, wird es als Leer-String interpretiert. Damit bei der Umwandlung in den ASCII-Code dadurch keine Fehlermeldung ausgelöst wird, wird zu A$ immer CHR$(0) addiert. Spulen Sie nun zurück und starten das Programm erneut. Auf Ihrem Bildschirm erscheint nun folgendes:

```
S83
T84
R82
I76
N78
G71
149
44
S83
.
.
.
N78
G71
452
13
0
32
.
.
.
```

Die Null nach STRING4 ist die Kennzeichnung für das Dateiende. Danach folgt nur noch "SPACE".

10.2 Die Sekundäradresse

Wie ich schon im ersten Unterkapitel geschrieben habe, gibt es neben der Geräteadresse auch noch eine Sekundäradresse. Diese Adresse wird allgemein dazu benutzt, dem Peripheriegerät oder dem Computer als Empfänger eine weitere Anweisung zur Betriebsart zu übermitteln.

Vorweg möchte ich noch ein paar grundlegende Bemerkungen über die zwei verschiedenen Programmarten machen. Zum einem gibt es BASIC-Programme. Diese Programme sind nicht abhän-

gig davon, in welchem Speicherbereich sie stehen, sie sind verschiebbar. Damit diese Programme auch immer an den Anfang des BASIC-Bereichs geladen werden, besitzt der C64 einen sog. "Relativlader", der automatisch alle BASIC-Programme an den BASIC-Start einliest.

Die andere Programmart ist das Maschinenprogramm. Diese Art von Programmen läuft nur in dem Speicherbereich, für den sie geschrieben wurde. Maschinenprogramme dürfen nicht relativ geladen werden. Um BASIC-Programme von Maschinenprogrammen unterscheiden zu können, wird die Sekundäradresse bei der Kassettenhandhabung benutzt. Im einzelnen bewirken die Sekundäradressen folgendes:

I. SAVE

0 Das abgespeicherte Programm wird als BASIC-Programm gekennzeichnet. Dadurch wird erreicht, daß dieses Programm mit dem Befehl LOAD automatisch an den gerade aktuellen BASIC-Start, also relativ, geladen wird.

1 Sie kennzeichnet ein Programm als Maschinenprogramm. Wenn Sie ein so gespeichertes Programm mit LOAD laden, wird es automatisch an die Adresse geladen, von welcher es abgespeichert wurde.

2 Sie speichert ein Programm als BASIC-Programm ab wie mit der Sekundäradresse 0. Zusätzlich schreibt der Rechner hinter das Programm noch einen sog. EOT (End Of Tape)-Block. Dieser Block sagt dem Rechner beim Lesen, daß er hier mit der Suche nach weiteren Programmen aufhören soll. Trifft der Computer beim Suchen eines Programms auf solch einen Block, ohne vorher ein Programm gefunden zu haben, so gibt er die Meldung ?FILE NOT FOUND ERROR aus.

3 Sie kennzeichnet ein Programm als Maschinenprogramm und schreibt einen EOT-Block dahinter.

II. LOAD

Bei dem Befehl LOAD hat die Sekundäradresse folgende Bedeutung:

1 Das Programm wird entsprechend der Information geladen, die im Programmkopf steht. D.h., als Maschinenprogramme gespeicherte Programme werden auch als solche geladen. Das gilt auch für BASIC-Programme, sie werden relativ geladen.
2 Jedes Programm, egal, wie es gekennzeichnet wurde, wird absolut, also an die Adresse geladen, von welcher es abgespeichert wurde.

Beim OPEN-Befehl gibt es folgende Sekundäradressen:

III. OPEN

0 Sie öffnet eine Datei zum Lesen.
1 Sie öffnet eine Datei zum Schreiben.
2 Sie öffnet eine Datei zum Schreiben und schreibt einen EOT-Block hinter die Datei.

10.3 Die Statusvariable

Wie ich schon in den vorausgegagenen Unterkapiteln erwähnt habe, gibt es in Ihrem Commodore eine festdefinierte Variable, die Ihnen Auskunft über den Verlauf einer Kassettenoperation gibt. Diese sog. "Statusvariable ST" wird vom Computer bei jedem Zugriff auf Kassette oder andere Peripherigeräte gesetzt. Mit ihr haben Sie die Möglichkeit, eventuell aufgetretene Fehler genauer zu spezifizieren. Weiterhin können Sie mit ihr feststellen, wann eine Datei zu Ende ist.

Diese Variable besteht aus acht einzelnen "Flags", die entsprechend den aufgetretenen Fehlern gesetzt werden. Mit Hilfe einer AND-Verknüpfung können Sie jedes einzelne Bit dieser Variable testen. Aus der Tabelle 1 können Sie die genaue Bedeutung der einzelnen Flags entnehmen.

Um das vierte Bit der Statusvariablen zu verstehen, muß man wissen, daß der Computer alle Daten und Programme zweimal auf Band schreibt und beim Einlesen beide Versionen miteinander vergleicht.

Tabelle 1

ST-Bit	ST-Dez.-	Bedeutung
		Äquivalent
0	1	Keine Bedeutung für die Kassette
1	2	--
2	4	Kurzer Block. Ein gefundener Block ist kürzer, als er sein müßte.
3	8	Langer Block. Ein gefundener Block ist länger, als er sein müßte.
4	16	Second-Pass-Fehler. Die Daten des ersten Pass stimmen nicht mit denen des zweiten überein.
5	32	Prüfsummenfehler. Die abgespeicherte Prüfsumme stimmt nicht mit der errechneten überein.
6	64	File-Ende
7	128	Bandende. EOT wurde gelesen.

Wie können Sie jetzt diese Variable in Ihren Programmen verwenden? Wie Sie aus der Tabelle entnehmen können, gibt das 6. Bit darüber Auskunft, ob eine Datei zu Ende ist. Wenn Sie also innerhalb Ihres Programms Bit sechs abfragen, können Sie das Einlesen nach dem letzten String beenden. Geben Sie einmal folgendes Programm ein, und lesen Sie damit die im Unterkapitel 10.1 angelegte Datei ein. Nachdem "STRING4" ausgedruckt wurde, erscheinen READY und der Cursor.

```
10 OPEN1,1,0,"TEST"
20 INPUT#1,A$
30 PRINT A$
40 IF(ST AND 64) = 64 THEN60
50 GOTO20
60 CLOSE1
70 END
```

10.3.1 Programme retten nach LOAD ERROR

Ihnen ist es bestimmt schon passiert, daß sich Ihr Computer nach dem Ladevorgang mit "?LOAD ERROR" meldete und das geladene Programm gar nicht oder nur teilweise eingelesen wurde. Die Ursache dafür ist meistens, daß die Position des

Tonkopfes relativ zum Band beim Schreiben eine andere war als beim Lesen. Abhilfe schafft da nur eine neue Justierung des Tonkopfes, wie sie im Unterkapitel 10.9.5 beschrieben wird.

Läßt sich das Programm nach neuer Tonkopfeinstellung immer noch nicht laden, liegt der Fehler am Band oder an einer fehlerhaften Speicherung des Programms. Sie brauchen aber nicht zu verzweifeln. Unter bestimmten Bedingungen ist es möglich, das Programm zumindest teilweise zu retten. Wenn Sie mit Ihrem Computer ein Programm abspeichern, wird es zweimal hintereinander auf Band geschrieben. Beim Einlesen wird die erste Version in den Speicher geladen und dann mit der zweiten verglichen. Stellt der Rechner dabei einen Fehler fest, versucht er erst diesen Fehler zu korrigieren. Ist das nicht möglich, setzt er Bit vier in der Statusvariablen und gibt ein ?LOAD ERROR aus.

Es kann auch sein, daß er einzelne Bits oder Byte-Kennzeichnung nicht lesen kann, was die Synchronisation zwischen Leseroutine und eingelesener Information "außer Tritt" bringt. Das hat zur Folge, daß die Bits zwei und/oder drei gesetzt werden. Jeder aufgetretene Fehler führt dazu, daß die BASIC-Vektoren 45 und 46 (Programmende) nicht entsprechend gesetzt werden.

Falls der Fehler erst gegen Ende des Programms oder erst im zweiten Pass aufgetreten ist, können Sie das Programm ganz oder zumindest teilweilse listen. Diesen listbaren Teil können Sie mit einem UNNEW-Befehl retten. Dies ist ein Befehl, der nicht im Commodore-BASIC 2.0 enthalten ist. Besitzen Sie keine BASIC-Erweiterung, die diesen Befehl enthält, können Sie das am Ende dieses Unterkapitels abgedruckte Programm eingeben. Falls Sie kein Monitorprogramm haben, geben Sie es mit dem BASIC-Lader ein und speichern es ab. Danach starten Sie den Lader. Nun können Sie das Maschinenprogramm von der angezeigten Startadresse bis zum BASIC-RAM-Ende abspeichern. Das so abgespeicherte Maschinenprogramm können Sie nun bei Bedarf einfach mit LOAD zuladen und mit SYS(Startadresse) starten, um entweder ein gelöschtes Programm wieder ins Leben zu rufen oder ein nur fehlerhaft ladbares Programm zumindest teilweise zu retten.

Nachdem Sie mit dem UNNEW-Befehl die Programmendevektoren gesetzt haben, speichern Sie es auf einer anderen Kassette ab. Nun kommt der schwierigste Teil. Jetzt müssen Sie feststellen, wo das Programm beschädigt ist. Das geschieht auf die gleiche Weise, wie Sie ein anderes BASIC-Programm auf Fehler untersuchen.

Wie schon anfangs bemerkt, ist der häufigste Grund für einen ?LOAD ERROR ein dejustierter Kopf. Darum sollten Sie als erstes immer versuchen, den Tonkopf besser auf das Band zu justieren und ggf. die oben beschriebene Prozedur mit verschiedenen Tonkopfeinstellungen zu wiederholen, bis Sie ein möglichst wenig beschädigtes Programm laden können.

```
100 REM UNNEW 64
110 PS=0:E=256*PEEK(55)-1:A=E-51
120 FOR I=A TO E:READ X:PS=PS+X:POKE I,X:NEXT
130 IF PS <> 5274 THEN PRINT"DATA ERROR":END
140 H=INT(A/256):POKE 56,H:POKE 55,A-256*H
150 PRINT"STARTADRESSE";A:NEW
160 DATA 165,43,164,44,133,34,132,35,160,3,200,177,34,208,251,200,152,24
,101
170 DATA 34,160,m0,145,43,165,35,105,0,200,145,43,32,51,165,165,34,105,2
,133
180 DATA 45,165,35,105,0,133,46,32,99,166,76,123,227
```

10.4 Laden und Speichern vom Programm aus

Bei der Behandlung der Befehle SAVE und LOAD im Unterkapitel 10.1 habe ich beschrieben, wie man Programme im Direktmodus abspeichert und lädt. Diese Befehle können aber auch innerhalb von Programmen stehen.

Der SAVE-Befehl arbeitet innerhalb von Programmen genauso wie im Direktmodus. Es ist somit möglich, ein Programm sich auch selber abspeichern zu lassen. Diese Möglichkeit werden Sie wahrscheinlich aber nur selten nutzen. Weit häufiger möchte man nur einen Teil eines BASIC-Programms oder ein Maschinenprogramm abspeichern.

Stellen Sie sich vor, Sie haben ein Programm geschrieben, das ab Zeile 10.000 Datazeilen generiert. Diesen Programmteil wollen Sie alleine abspeichern, um ihn dann an andere Programme anhängen zu können. Sie müssen als erstes feststellen, ab welcher Speicherstelle die Zeile 10.000 im Speicher steht, da Sie ja nur ab Zeile 10.000 abspeichern wollen. Mit den folgenden Programmzeilen lösen Sie dieses Problem:

```
1000 S = PEEK(43) + 256 * PEEK(44) : BS = S
1010 ZN = PEEK(S + 2) + 256 * PEEK(S + 3)
1020 IF ZN = 10000 THEN 1050
1030 S = PEEK(S) + 256 * PEEK(S + 1)
1040 GOTO 1010
```

Um diese Programmzeilen zu verstehen, muß ich Ihnen kurz erklären, wie BASIC-Zeilen im Speicher stehen. Die ersten beiden Bytes einer BASIC-Zeile bildet die sogenannte Koppeladresse. Sie gibt die Startadresse der nächsten BASIC-Zeile an, zeigt also auf die nächste Koppeladresse. In den beiden folgenden Bytes ist die Zeilennummer abgespeichert. Danach kommt der Inhalt der BASIC-Zeile, abgeschlossen durch ein Null-Byte. Ihr BASIC-Programm steht also folgendermaßen im Speicher:

```
KAL KAH ZNL ZNH BASICtext 00 KAL KAH ZNL ZNH ....
```

Hierbei bedeuten:

```
KAL  Koppeladresse LSB
KAH  Koppeladresse MSB
ZNL  Zeilennummer LSB
ZNH  Zeilennummer MSB
```

Bei unserem kleinen Programm wird in der Zeile 1.000 die Programmstart-Adresse aus den Vektoren in S und BS eingelesen. In der Zeile 1.010 wird die Zeilennummer in die Variable ZN übertragen. Die Abfrage in der Zeile 1.020 testet, ob die gewünschte Zeilennummer erreicht wurde. Wenn ja, wird in das weitere Programm verzweigt. Ist sie noch nicht erreicht, wird in Zeile 1.030 aus der Koppeladresse die Startposition der nächsten BASIC-Zeile in S geschrieben und über Zeile 1.040 wieder nach 1.010 gesprungen.

Nach diesem Programm haben Sie alle Informationen, um den letzten Teil Ihres Programms abzuspeichern. Wichtig ist nun nur noch, daß Sie nach dem Speichervorgang wieder den alten Wert in den BASIC-Start-Vektor schreiben. Mit dem folgenden Programmteil können Sie die Datazeilen abspeichern.

```
1050 POKE 43,S AND 255: POKE 44,INT(S/256) :REM ZEIGER
AUF ZEILE 10000 SETZEN
1060 SAVE"DATAZEILEN" :REM DATAZEILEN SPEICHERN
1070 POKE 43,BS AND 255: POKE 44,INT(BS/256) :REM ZEIGER ZURUECKSETZEN
```

10.4.1 Overlay-Technik

Der LOAD-Befehl arbeitet innerhalb von Programmen etwas anders als im Direktmodus. Im Unterkapitel 10.1 habe ich geschrieben, daß nach einem LOAD die BASIC-Zeiger auf das neue Programm eingestellt werden. Geben Sie aber die LOAD-Anweisung innnerhalb von Programmen, bleiben die Vektoren ebenso erhalten wie alle bis dahin definierten Variablen.

Daher ist es möglich, von einem Hauptprogramm aus verschiedene Unterprogramme aufzurufen, die alle mit den gleichen Variablen arbeiten. Die Methode nennt man "Overlay-Technik".

Bei dieser Technik sind aber einige Dinge zu beachten:

1. Da die BASIC-Vektoren nicht verändert werden, muß das aufrufende Programm mindestens so lang sein wie das aufgerufene.

   Direkt hinter einem BASIC-Programm werden die Variablen abgespeichert, und der Vektor 45, 46 ist damit auch der Zeiger für den Beginn der Variablentabelle. Ist nun das nachgeladene Programm länger, werden die Variablen überschrieben, und der Zeiger weist in das neue Programm. Sobald nun eine Variablenoperation stattfindet, wird die Variable innerhalb des Programms gesucht, was zum "Absturz" des Rechners führen kann.

Mit einem Trick ist es aber möglich, von einem kurzen Programm aus ein längeres aufzurufen.

Zuerst laden Sie das längste nachzuladende Programm im Direktmodus ein und stellen durch Auslesen der Speicherstellen 45,46 seine Länge fest. Die erhaltenen Werte notieren Sie sich am besten. Als nächstes laden Sie das aufrufende Programm und setzen folgende BASIC-Zeile an den Anfang: 10 POKE 45,W1: POKE 46,W2: CLR.

Die Zeilennummer ist wahlweise, sie muß nur die kleinste in Ihrem Programm sein. W1 und W2 stehen für die von Ihnen notierten Werte.

Durch die POKE-Befehle verlängern Sie Ihr Programm künstlich. Das CLR bewirkt, daß auch die anderen Zeiger entsprechend geändert werden.

2. Sie müssen beachten, daß nach einem LOAD das Programm wieder von Anfang an ausgeführt wird.

   Das ist nur dann sinnvoll und unproblematisch, wenn Sie BASIC-Programme nachladen. Wollen Sie aber ein Maschinenprogramm nachladen, ändert sich das im Speicher befindliche BASIC-Programm nicht und startet sich immer wieder von neuem. Wenn das nicht geschehen soll, müssen Sie das durch eine Sprungtabelle verhindern. Bei dem folgenden Programm werden am Anfang des Programms drei Maschinenprogramme eingeladen, ehe es zum Hauptprogramm übergeht.

```
10 REM START
20 IF A = 0 THEN A=1: LOAD"MPR1",1,1
30 IF A = 1 THEN A=2: LOAD"MPR2",1,1
40 IF A = 2 THEN A=3: LOAD"MPR3",1,1
50 REM HAUPTPROGRAMM3
```

3. Als letztes ist eine Besonderheit der String-Speicherung zu beachten.

   Ihr Commodore-Betriebssystem ist so konzipiert, daß möglichst wenig Speicherplatz für Strings verwendet wird. Je-

desmal, wenn Sie einer String-Variablen einen String zuweisen, wird in der Variablentabelle unter dem Variablennamen ein Zeiger abgelegt, der auf den betreffenden String zeigt. Weisen Sie einer Variablen durch eine INPUT-Anweisung einen String zu, wird dieser in einer speziellen Tabelle am Ende des BASIC-RAMs abgelegt. Geschieht die Zuweisung aber innerhalb des Programms, z.B. durch

```
100 A$="STRING"
```

wird der String nicht noch einmal in die String-Tabelle übertragen, sondern der Zeiger wird auf die Position des Strings innerhalb des Programms gesetzt.

Laden Sie nun ein Programm nach, steht an der entsprechenden Stelle etwas ganz anderes und der Computer druckt bei dem Befehl PRINT A$ nur wirres Zeug aus. Abhilfe schaffen Sie sich dadurch, daß Sie scheinbar einen neuen String definieren, indem Sie den im Programm definierten mit einem Leer-String verknüpfen. Schreiben Sie also:

```
100 A$="STRING": A$=A$+""
```

Dadurch bringen Sie das Betriebssystem dazu, den String in die String-Tabelle zu übertragen.

10.5 Der Kassettenpuffer

In diesem Unterkapitel möchte ich näher auf den Kassettenpuffer eingehen. Dieser Puffer belegt den Speicherbereich von 828 (Hexadezimal 033c) bis 1019 (Hexadezimal 03FB). Dieser Puffer hat 2 grundsätzliche Aufgaben. Zum einen wird in diesem Bereich der File-Kopf, der sogenannte "Header", generiert und vor jedem File, sei es ein Programm- oder Daten-File, abgespeichert und dient dann zur Kennzeichnung desselben. Er enthält dann folgende Parameter:

1. File-Typ,Byte 0 = 828
2. Startadresse, Byte 1, 2 = 829, 830
3. Endadresse,Byte 3, 4 = 831, 832
4. File-Namen ab Byte 5 = 833

Die restlichen Bytes sind mit Space beschrieben. Als zweite Funktion dient der Kassettenpuffer als Zwischenspeicher bei der Datensicherung. Die abzuspeichernden Daten werden zuerst in diesen Speicherbereich geschrieben, und erst, wenn der ganze Puffer beschrieben wurde, wird er auf Band übertragen. Wenn Sie Daten laden, werden diese blockweise geladen und dann durch INPUT# oder GET# aus dem Puffer gelesen, wie auch im Unterkapitel 10.1.4 ff beschrieben.

Der File-Typ ist folgendermaßen codiert:

```
1 = BASIC-Programm, relativ laden
2 = Datenblock
3 = Maschinenprogramm, absolut laden
4 = Daten-Header
5 = EOT-Block
```

Im Header sind alle wichtigen Daten enthalten, die zur Kennzeichnung benötigt werden. Was liegt da näher, als mit Hilfe dieser Daten die mit Programmen und Daten bespielten Kassetten zu archivieren.

10.5.1 Anlegen eines Kassetten-Inhaltsverzeichnisses

Das Programm am Ende dieses Unterkapitels gibt ein Inhaltsverzeichnis einer Kassette aus. Falls Sie einen Drucker haben, können Sie sich das Inhaltsverzeichnis auch ausdrucken lassen.

Programmbeschreibung

Die Zeilen 100 - 160 dienen zur Initialisierung. Hier werden die einzelnen Tabulatorstops in Tx$ und die verschiedenen File-Typen in das Feld TN$ eingelesen. Für den Fall, daß die Ausgabe auch zu einem Drucker geschickt werden soll, wird in Zeile 180 ein Druckerkanal eröffnet und in 190 - 220 die Überschrift an

den Drucker übermittelt. In der Zeile 230 wird dann eine Datei zum Kassettenlesen ohne Namen eröffnet. Dadurch wird der nächste Header eingelesen.

Durch PEEK werden in den Zeilen 240 - 270 Typ-Byte, Start- und Endadresse und File-Name aus dem Puffer gelesen und formatiert. In den Zeilen 280 - 290 wird aus Start- und Endadressen die Programmlänge in KByte berechnet.

Durch die Zeilen 300 - 320 werden die Daten auf Drucker und Bildschirm ausgegeben. In Zeile 330 wird über die Programmlängen der Zählerstand berechnet. Nachdem die Datei geschlossen wurde, springt das Programm wieder zum OPEN-Befehl.

```
100 REM                    *INHALTSVERZEICHNIS***
110 PA=828:REM STARTADRESSE IM CASSETTENPUFFER
120 PRINT"DRUCKEN ? [J/N]  ";D$
130 GETD$:IFD$=""THEN130
140 T1$=CHR$(16)+"05":T2$=CHR$(16)+"10":T3$=CHR$(16)+"20":T4$=CHR$(16)+"
30"
150 T5$=CHR$(16)+"38":T6$=CHR$(16)+"50":T7$=CHR$(16)+"63":REM TABULATOR-
STOPPS
160 TN$(0)="RELATIV":TN$(2)="ABSOLUT":TN$(3)="DATEI"
170 IFD$<>"J"THEN230
180 OPEN2,4
190 PRINT#2,T1$"LFN"T2$"ZAEHLER"T3$"TYP"T4$"K-BYTE"T5$"   ANFANG"T6$"
ENDE";
200 PRINT#2,T7$"  NAME"
220 PRINT#2
230 OPEN1,1
240 TY=PEEK(PA):TY$=TN$(TY-1)
250 A$=" :"+RIGHT$("     "+STR$(PEEK(PA+1)+(256*PEEK(PA+2))-1),6)
260 B$=" -"+RIGHT$("     "+STR$(PEEK(PA+3)+(256*PEEK(PA+4))-1),6)
270 C$=": ":FORI=5TO20:C$=C$+CHR$(PEEK(PA+I)):NEXT
280 T=VAL(RIGHT$(B$,6))-VAL(RIGHT$(A$,6))
290 K$=RIGHT$("     "+STR$(INT(T/1024*100)/100),6)
300 PRINTZ;TY$;K$;A$B$:PRINTC$:PRINT
310 N=N+1:N$=RIGHT$("      "+STR$(N),3)
315 IFD$<>"J"THEN330
320 PRINT#2,T1$;N$; T2$;Z; T3$;TY$; T4$;K$; T5$;A$; T6$;B$; T7$;C$
330 Z=INT(Z+(T/212)+4+2*<PI>*(N/100)):REM BANDZAEHLER BERECHNEN
340 CLOSE1
350 GOTO230
```

10.5.2 Anzeige der gefundenen Dateien

Das Auslesen des File-Namens mittels eines PEEK-Befehls kann auch innerhalb eines Dateiverwaltungsprogramms nützlich sein. Haben Sie sich noch nicht darüber geärgert, daß, wenn Sie von einem Programm eine bestimmte Datei einlesen wollen, der Computer ewige Zeit sucht, um schließlich ?FILE NOT FOUND auszugeben. Der Grund dafür war vielleicht, daß Sie das Band etwas zu weit vorgespult hatten.

Wenn der Rechner Ihnen aber sagt, welche Dateien er findet, können Sie leicht feststellen, ob Sie zu weit gespult haben oder nicht. Vorausgesetzt natürlich, daß Sie die ungefähre Reihenfolge der Dateien auf Band kennen. Die folgenden BASIC-Zeilen veranlassen den Computer, jede Datei, die er findet, anzuzeigen.

```
100 INPUT"FILE-NAME";F$
110 L=LEN(F$)
120 FE$=""
130 OPEN4,1,0,FE$
140 FORI=0TO15:FE$=FE$+CHR$(PEEK(833+I)):NEXT
150 PRINT:PRINT"FOUND ";FE$
160 IFLEFT$(FE$,L)<>LEFT$(F$,L)THENCLOSE4:GOSUB200:GOTO120
170 PRINT:PRINT"FILE WIRD GELADEN"
200 POKE198,0:WAIT198,1:RETURN
```

Durch den OPEN-Befehl ohne Namen in Zeile 130, veranlassen Sie den Rechner, jeden Datenkopf in den Puffer zu laden. In Zeile 140-160 wird dann der File-Name mittels PEEK aus dem Puffer in FE$ übertragen und in Zeile 170 angezeigt. Ergibt der Vergleich von FE$ mit dem gewünschten File-Namen in Zeile 180 keine Übereinstimmung, wird die Suche fortgesetzt. Andernfalls wird die Datei geladen.

10.5.3 Selbststartende Programme und Programmschutz

Bis jetzt haben wir nur Daten aus dem Kassettenpuffer ausgelesen. Dieser Puffer eignet sich aber auch vortrefflich, um Daten

oder Programme aufzunehmen. So ist es möglich, durch diesen Puffer noch weitere Daten oder ein Programm mit einem anderen Programm abzuspeichern.

Der Puffer belegt den Speicherbereich von 828 bis 1.019. Er ist also 191 Zeichen lang. Zur Informationsübergabe werden, wenn er als Header abgespeichert wird, maximal 16 + 2 + 2 + 1 (File-Name, Startadresse, Endadresse, Typ-Byte) = 21 Byte benötigt. Es bleiben also noch 170 Byte übrig, ausreichend Platz, um kleine Maschinenprogramme unterzubringen.

Als Beispiel beschreibe ich Ihnen hier ein kleines Programm, das ein Vorprogramm erzeugt, das Ihren Wünschen entsprechend z.B. das nächste Programm automatisch lädt und startet.

Solche Vorprogramme können auch dazu dienen, Ihre Programme zu schützen. Ebenfalls kann mit solch einem Vorprogramm erreicht werden, daß vor dem Laden des Hauptprogramms der BASIC-RAM verschoben wird, um vor das BASIC-Programm noch eine oder mehrere Grafikseiten abzulegen. Dem Programmierer sind hier keine Grenzen gesetzt. Doch nun zu unserem Beispiel. Ich habe dieses Beispiel gewählt, da es einerseits die allgemeine Technik zeigt, und andererseits auch von einem "Nur-Anwender" recht allgemein genutzt werden kann. Hier ist die Programmbeschreibung:

In Zeile 10 wird der File-Name abgefragt, unter welchem das Hauptprogramm abgespeichert werden soll. In Zeile 20 und 30 wird der File-Name durch Auffüllen mit "SPACE" auf eine Länge von 16 Zeichen gebracht. Durch die Zeilen 40 bis 70 wird an den File-Namen ein Maschinenprogramm angehängt, welches in den Datazeilen ab Zeile 180 steht.

Über die Zeilen 80 und 90 haben Sie nun die Möglichkeit, BASIC-Befehle einzugeben, die später nach dem Laden des Vorprogramms automatisch ausgeführt werden. Wenn Sie also wünschen, daß ein folgendes BASIC-Programm geladen und gestartet werden soll, geben Sie LOAD <Return> RUN <Return> <Return> in Kurzbefehlen ein:

```
lO <Return> rU <Return> <Return>
```

Aus programmtechnischen Gründen, die ich später beschreiben werde, ist es bei diesem Programm nicht möglich, mehr als 10 Zeichen zu übergeben. Diese Zeichen werden dann an den File-Namen hinter das Maschinenprogramm angehängt (Zeile 100 - 150). In Zeile 160 wird der Zeiger zu einer Systemroutine, der Eingabe-Warteschleife, auf das Maschinenprogramm im Kassettenpuffer umgesetzt.

Die Zeile 170 speichert dann mit Hilfe des im letzten Unterkapitel beschriebenen Maschinenprogramms, diesen Vektor ab. Sie müssen also hinter den SYS-Befehl die Startadresse schreiben, die Ihr SAVE-ADRESS-Programm hat. Durch diesen Speicherbefehl wird der File-Name mit Maschinenprogramm und BASIC-Befehlen in den Puffer und auf Band geschrieben. Nach dem Programmende springt der Computer über den Vektor in 770, 771 normalerweise in die Eingabe-Warteschleife. Da dieser aber in Zeile 160 auf unser Maschinenprogramm im Kassettenpuffer gesetzt wurde, springt er dorthin und arbeitet es ab. Da wir im Ausgabe-String dem Rechner den Befehl LOAD gegeben haben, versucht er das nächste Programm zu laden. Durch das Drücken der <Run/Stop>-Taste kann man diesen Vorgang unterbrechen, um das Hauptprogramm hinter das Ladeprogramm zu speichern, das vom Vorprogramm geladen werden soll.

Wenn Sie ein Programm hinter den Lader geschrieben haben, spulen Sie das Band zurück zum Start des Ladeprogramms und geben LOAD und <Return> ein. Ihr Computer findet nun das Ladeprogramm, lädt dieses, lädt sofort danach das folgende Programm ein und startet es. Für Interessierte und Fortgeschrittene möchte ich nun diese Technik und das Programm genau erklären. Die zwei Grundlagen dieser Technik sind folgende:

1. Übergabe eines Maschinenprogramms in den Kassettenpuffer durch Anhängen an den File-Namen,
2. Änderung von Vektoren, z.B. dem zur Eingabeschleife auf ein eigenes Maschinenprogramm, und Speicherung eben dieser geänderten Vektoren. Im eigenen Maschinenprogramm muß dann vor dem Sprung in das geladene Programm der in

das eigene Maschinenprogramm zeigende Vektor wieder auf den alten Wert gebracht werden.
3. Als Maschinenprogramme abgespeicherte Speicherbereiche werden immer an die Stelle geladen, von wo sie abgespeichert wurden.

Ihr Rechner besitzt in den Speicherstellen 768 - 819 eine Reihe von Vektoren, über welche er in bestimmte Unterprogramme springt. Durch Veränderung dieser Zeiger können Sie ein eigenes Programm dazwischenschalten. Die Eingabe-Warteschleife ist das Unterprogramm, das Ihren Commodore geduldig auf irgendeine Eingabe warten läßt. Machen Sie eine Eingabe, wertet und führt er sie aus, um anschließend wieder in diese Schleife zurückzukehren.

Durch unser Programm wird nun ein Vorprogramm erzeugt, das den Vektor zur Eingabe-Warteschleife mit der Startadresse eines eigenen Programms überschreibt, das mit dem File-Namen in den Speicher geschrieben wurde. Wenn Sie selber weitere Experimente machen wollen, können Sie natürlich auch Ihr Maschinenprogramm z.B. in den Bereich $2CO - $2FF (704 - 766) legen und als Vorprogramm den Bereich $2C0 - 303 abspeichern.

Doch sehen wir uns nun das Maschinenprogramm an. In den Zeilen 200 - 230 wird der Vektor zur Eingabe-Warteschleife wiederhergestellt. Ab der Speicherstelle $036C sind Ihre Eingaben abgespeichert. Diese werden in den Zeilen 250 bis 290 in den Tastaturpuffer übertragen. Durch den Sprung am Ende des Programms in die Eingabe-Warteschleife wird der Rechner veranlaßt, die Befehle, die im Tastaturpuffer stehen, auszuführen.

Das hier dargestellte Beispiel ist zwar recht einfach, hat aber den Nachteil, daß Sie nur wenige Befehle aufgrund des begrenzten Tastaturpuffers übergeben können. Sie können jetzt aber, wenn Sie das Prinzip verstanden haben, leicht dieses Programm auf Ihre eigenen Bedürfnisse hin abändern, indem Sie im Maschinenprogramm ab Zeile 240 Ihre eigene Routine schreiben, und diese mit einem Sprung in die Eingabe-Warteschleife abschließen. Ihnen ist bestimmt schon klar geworden, daß man diese Technik auch sehr gut zum Programmschutz verwenden

kann, da sich das Programm ja selbst startet. Wenn Sie zusätzlich noch einen Programmabbruch durch die <Run/Stop>-Taste unterbinden und das Hauptprogramm so abändern, daß es nur in Verbindung mit dem Lader lauffähig ist, hat ein fremder Anwender keine Möglichkeit, Einsicht in Ihr Programm zu nehmen.

Der ganze Schutz wäre aber nutzlos, wenn ich hier eine genaue Anleitung geben würde. Ich hoffe, daß Sie nach meinen Ausführungen und mit etwas Maschinenprogrammierkenntnis selbst in der Lage sind, eigene Autostart-Routinen zu entwickeln.

```
5 REM LADER FUER C-64
10 INPUT"FILE-NAME ";F$
20 SP$="                "
30 F$=LEFT$(F$+SP$,16):S=833
40 FORI=0TO26
50 :  READF
60 F$=F$+CHR$(F)
70 NEXT
80 PRINT"AUSGABESTRING MAX. 10 ZEICHEN "
90 INPUT"^ = RETURN, AM ENDE 2 MAL";A$
100 LA=LEN(A$):IFLA>10THEN90
110 F$=F$+CHR$(LA)
120 FORI=1TOLA
130 :  W$=MID$(A$,I,1):IFW$="^"THENW$=CHR$(13)
140 :  F$=F$+W$
150 NEXT
160 POKE770,81:POKE771,3
170 SYS12*4096F$,1,768,772
180 DATA169,131,141,2,3,169,164,141,3,3,174,108,3,134,198,189,108,3,157,
119
190 DATA2,202,208,247,76,131,164
```

10.6 Speicherformat der Kassettenspeicherung

Um Daten oder Programme auf Band aufzuzeichnen, werden einzelne Bytes und Bits als Pulse aufgezeichnet. Zur Codierung sind drei verschiedene Zeiten definiert:

```
K=kur (176 Microsekunden)
M=mittel (256 Microsekunden)
L=lang (336 Microsekunden)
```

Aus diesen Pulsen werden drei verschiedene Kombinationen gebildet, die die folgende Bedeutung haben:

```
LLMM=Byte, diese Kombination geht jedem Byte voraus
MMKK =Bit gesetzt = 1
KKMM =Bit nicht gesetzt = 0
```

Der Wert 65 wird also wie folgt auf Band aufgezeichnet:

```
LLMM MMKK KKMM KKMM KKMM KKMM KKMM MMKK KKMM MMKK
Byte   1  0  0  0  0  0  1  0  1
Bit    0  1  2  3  4  5  6  7  PARITY ODD
```

Dies ergibt eine Zeitspanne von 8.96 ms für ein Zeichen. Beim Lesevorgang zählt ein Zähler von einen bestimmten Wert so lange abwärts, wie ein Signal an der Leseleitung des Computers liegt. An dem erreichten Werten erkennt dann der Rechner, ob es sich um ein gesetztes Bit, ein nicht gesetztes Bit oder eine Byte-Marke handelt.

Damit der Zähler aber immer zur richtigen Zeit gestartet wird, ist eine genaue Synchronisierung zwischen Band und Zähler notwendig. Aus diesem Grunde geht jedem Block eine Synchronisation und ein Countdown voraus.

Ein Block ist folgendermaßen aufgebaut:

1. Synchronisations-Bytes und Countdown
   Durch diesen Teil werden die Lesebausteine auf das Band synchronisiert. Weiterhin enthält die Synchronisation auch die Informationen, um was für einen Block es sich handelt.
2. Daten
3. Prüfsummen-Byte
   Während des Abspeicherns und Ladens bildet der Rechner aus allen Bytes eine EXOR-Prüfsumme, indem er den letzten Wert mit dem nächsten durch eine logische "Exclusive-Oder-Funktion" verknüpft und abspeichert. Wenn Lesefehler auftreten, stimmt diese Prüfsumme nicht mit der abgespeicherten überein.
4. Wiederholung der Daten
   Diese Wiederholung wird beim Lesevorgang mit den schon eingelesenen Bytes verglichen, um eventuelle Lesefehler festzustellen.

5. Prüfsummen-Byte (wie 3)
6. Blockendemarkierung

Die kompletten Files werden wie folgt abgespeichert:

	Programm-Files	Daten-Files
1.	Header	Header
2.	Programm	Datenblock (evtl. weitere Datenblöcke)
3.	evtl. EOT-Block	evtl. EOT-Block

10.7 Append von BASIC-Programmen

Sie haben sich bestimmt schon darüber geärgert, daß das Commodore BASIC 2.0 keine Befehle zur Verfügung stellt, um mehrere Programme miteinander im Speicher zu verknüpfen. Wieder einmal kommen uns die BASIC-Zeiger 43, 44 / 45, 46 zu Hilfe. Sie "verbiegen" den BASIC-RAM-Startzeiger auf das Ende des im Speicher befindlichen Programms und laden das anzuhängende Programm nach. Anschließend muß der Zeiger 43, 44 wieder auf den alten Wert gebracht werden.

Im einzelnen müssen Sie wie folgt vorgehen:

1. Laden Sie das erste Programm ein. Die Zeilennummern müssen alle kleiner sein als die des anzuhängenden Programms. Sind sie es nicht, arbeitet das zusammengefügte Programm nicht so, wie es eigentlich sollte.

2. Stellen Sie durch

```
PRINT PEEK(43),PEEK(44)
```

   die BASIC-Startadresse fest, und notieren Sie sie am besten.

3. Geben Sie nun

```
POKE43,(PEEK(45)+256*PEEK(46)-2)And255
POKE44,(PEEK(45)+256*PEEK(46)-2)/256
```

ein. Da am Ende eines BASIC-Programms immer zwei Null-Bytes zur Endemarkierung stehen, müssen Sie den Endevektor um zwei Bytes vermindern. Nun haben Sie scheinbar kein Programm mehr im Speicher. Dies können Sie überprüfen, wenn Sie LIST eingeben.

4. Laden Sie nun das zweite Programm ein. Wenn Sie jetzt LIST eingeben, erscheint nur das zweite Programm.

5. Schreiben Sie nun die alten Werte wieder in die Speicherstellen 43, 44. Durch diese Prozedur haben Sie nun zwei Programme aneinandergefügt.

Eleganter geht es natürlich mit einem Maschinenprogramm. Das folgende kleine Programm erkennt durch ein " z!", dem Klammeraffen, als erstes Zeichen im File-Namen, daß das zu ladene Programm an das im Speicher befindliche angehängt werden soll.

Programmbeschreibung

Durch die Initialisierung wird der Ladevektor auf dieses Programm verschoben. Dadurch wird bei jeder Ladeoperation zu diesem Programm verzweigt. In diesem Programm wird dann das erste Zeichen des File-Namens auf " z!" getestet. Findet das Programm dieses Zeichen nicht, springt es in die alte Routine. Wird dieses Zeichen gefunden, wird in den Lade-Startadressenvektor $C3/$C4 der Programmendevektor minus 2 übertragen und das Zeichen " " im File-Namen gelöscht. Danach springt das Programm wieder in die alte Laderoutine.

```
10 REM MERGE C64
20 E=256*PEEK(56)+PEEK(55)-1:A=E-41
30 FORI=ATOE:READX:POKEI,X:NEXT
40 READX,Y,Z:X=A+X:Y=A+Y:Z=A+Z:H=INT(Z/256)
45 POKEX,Z-256*H:POKEY,H
46 POKEE-1,PEEK(816):POKEE,PEEK(817)
50 H=INT(A/256):POKE56,H:POKE55,A-256*H
65 SYSA:NEW
32000 DATA169,224,162,127,141,48,3,142,49,3,96,72,162,0,161,187,201,64,2
08,18
32001 DATA56,165,45,233,2,133,195,165,46,233,0,133,196,24,230,187,198,18
3,104
32002 DATA76,165,244,1,3,11
```

10.8 Steuerung der Datasette per Programm

Bis jetzt habe ich nur über den Lade- und Speichervorgang mit Ihrer Datasette geschrieben. Dabei haben wir gesehen, daß das Band automatisch gestartet und gestoppt wurde. Der Computer besitzt also die Möglichkeit, den Motor zu steuern. Unter dem Motto, "was der Computer kann, kann ich schon lange", werde ich Ihnen jetzt beschreiben, wie Sie diese Steuerungsmöglichkeit in Ihrem Programm nutzen können.

Mit folgenden Speicherstellen können Sie den Motor steuern und abfragen, ob eine Taste am Rekorder gedrückt wurde:

Adr.	Wert	Funktion
1	AND 223	Motor an
1	OR 32	Motor aus
192	0	Motor an
192	1	Motor aus
1	16	Taste gedrückt

Auf eine Taste am Rekorder warten:

```
WAIT 1,16,16
```

Als Anwendungsbeispiel, wie Sie die Steuerungsmöglichkeiten nutzen können, folgt ein Programm, das Ihnen das Suchen eines Programms, das unter vielen auf einer Kassette abgespeichert ist, abnimmt. Es erstellt einen Katalog aller auf Ihrer Kassette gespeicherten Programme und findet ein bestimmtes Programm selbständig. Das Programm arbeitet nach folgendem Prinzip:

Nachdem Sie ein oder mehrere Programme hinter dieses Beispielprogramm auf Kassette abgespeichert haben, stellt es mit Ihrer Hilfe und der Variablen "TI" die Zeit fest, die der Kassettenrekorder benötigt, um vom Ende des Katalogprogramms bis zum zu katalogisierenden Programm vorzuspulen. Dieser Zeitwert wird mit dem File-Namen des Programms in Datazeilen abgespeichert. Soll später ein bestimmtes Programm gesucht werden, wird der Kassettenmotor beim Vorspulen so lange angelas-

sen, bis der aktuelle Zeitwert mit dem abgespeicherten übereinstimmt. Nach Drücken der <Play>-Taste wird dann das gewünschte File geladen. So bedienen Sie das Programm:

Dieses Programm wird am Anfang der Kassette abgespeichert. Mit einem Abstand von ca. 10 Sekunden können Sie dahinter wie gewohnt die anderen Programme abspeichern. Sollen die Programme in den Katalog aufgenommen werden, notieren Sie sich die Startzähler-Nummern und File-Namen der einzelnen Programme. Anschließend spulen Sie die Kassette an den Start zurück und laden das Katalogprogramm. Nach Starten des Programms wählen Sie den Punkt "Neueingabe".

Zuerst werden Sie gebeten, die entsprechenden File-Namen in der richtigen Reihenfolge einzugeben. Sie werden dann vom Programm aufgefordert, den schnellen Vorlauf am Rekorder einzustellen. Nun können Sie den Rekorder durch Drücken der <Space>-Taste starten und bei den notierten Zählerständen genauso stoppen, wieder starten und beim nächsten File-Start stoppen, bis Sie so allen eingegebenen Programmen einen Zeitwert zugeordnet haben. Als letztes speichern Sie das Katalogprogramm wieder am Kassettenanfang ab. Nun können Sie komfortabel ein im Katalog gespeichertes Programm nach der Angabe "LADEN" aus einem Bildschirmmenü wählen. Bei gedrückter FFWD spult der Rechner bis zum gewünschten Programm. Nach Drücken der <Play>- und <Space>-Taste wird das Programm geladen.

Programmbeschreibung

In den Zeilen 120 bis 170 sind die Unterprogramme zum Starten und Stoppen des Motors. In den Zeilen 180 bis 340 wird das Programm initialisiert und das Menü auf den Bildschirm geschrieben. Bei der Initialisierung werden die Namens- und Zeitfelder dimensioniert und die Repeat-Funktion für alle Tasten eingeschaltet. Mit Hilfe der Betriebssystem-Routine SCREEN (65517) wird der Computertyp festgestellt und dementsprechend der Start vom Video- und Farb-Ram gesetzt. Zuletzt werden die Variablen zur Motorsteuerung entsprechend

dem Computer belegt (Zeilen 240 und 260). Aufgrund der Eingabe wird zur "Neueingabe" oder zum "Laden" verzweigt.

In den Zeilen 350 bis 660 befindet sich der Block "Neueingabe". In der Zeile 390 wird die Anzahl der schon eingegebenen Files über READ aus den Datazeilen eingelesen. In den Zeilen 400 - 420 werden die neuen File-Namen eingelesen und im Feld N$(n) abgespeichert.

In den Zeilen 530 bis 580 wird die Spulzeit zwischen den Startpunkten der einzelnen Programmen festgestellt und, den eingegebenen File-Namen entsprechend zugeordnet, im Feld ZS(n) gespeichert. Vorher wird in der Zeile 580 ein "Offset" von 10 abgezogen, um die Auslaufzeit des Motors auszugleichen. Aus gleichem Grunde wird bei der Summation der einzelnen Differenzzeiten der Zeitwert 10 addiert. Um Fehlbedienungen zu vermeiden wird in Zeile 420 getestet, ob eine Taste am Rekorder gedrückt ist, und der Anwender ggf. gebeten die <Stop>-Taste zu betätigen.

In den Zeilen 600 - 660 werden File-Namen und Zeiten in Datazeilen geschrieben und das Programm beendet. In Zeile 670 beginnt der Programmpunkt "Laden". Die Datenanzahl und die File-Daten werden eingelesen, und in den Zeilen 720 - 820 werden die Files, je 8 Stück pro Seite, auf dem Bildschirm mit einem " zä", das über die Zeilen 770 - 810 gesteuert wird, ausgedruckt. Ist die Abfrage nach der Taste F7 in Zeile 790 positiv, wird nach Zeile 830 zur Such- und Laderoutine verzweigt.

Nach dem Test, ob eine Taste am Rekorder gedrückt ist (840), wird in der Zeile 900 mit dem WAIT-Befehl auf das Drücken der <F.FWD>-Taste am Rekorder gewartet und dann, entsprechend der abgespeicherten Zeit der Kassettenmotor angelassen (Zeile 920 - 940). In den Zeilen 950 - 1.010 wird das gewünschte Programm geladen. Würde das Programm vom Programmodus her geladen, könnten nur Programme geladen werden, die kleiner als das Katalogprogramm sind. Um das zu vermeiden, wird der LOAD-Befehl bzw. der FastTape LOAD-Befehl L auf den Bildschirm und <Return> in den Tastaturpuffer

geschrieben. Somit wird das Programm aus dem Direktmodus geladen und die BASIC-Vektoren entsprechend dem zu ladenden Programm gesetzt.

```
10 GOTO190
20 ************************************
30 *                                  *
40 *           K A T A L O G          *
50 *                                  *
60 *           FUER CASSETTE          *
70 *                                  *
80 ************************************
90 :REM
100 REM          WARTESCHLEIFE
110 POKE198,0:WAIT198,1:GETA$:RETURN
120 POKE192,0:POKE1,PEEK(1)AND223:REM MOTOR AN C-64
130 :REM POKE 37148,252 :REM MOTOR AN VC20
140 RETURN
150 POKE192,1:POKE1,PEEK(1)OR32:REM MOTOR AUS C-64
160 :REM POKE 37148,0 :REM MOTOR AUS VC20
170 RETURN
180 REM "         INITIALISIERUNG
190 HD$=" * * K A T A L O G * *"
200 SYS65517:SP=PEEK(781):ZE=PEEK(782):REM BILDSCHIRMDATEN HOLEN
210 BS=1024:BF=55296:FW=1:IFSP>25THEN260
220 BS=4*(PEEK(36866)AND128)+64*(PEEK(36869)AND120)
230 BF=37888+4*(PEEK(36866)AND128)
240 WZ=37151:WW=64
250 FW=6:GOTO270
260 WZ=1:WW=16:REM C-64 WERTE 'WARTEN AUF RECORDERTASTE
270 DIMN$(50),ZS(50):REM NAME UND STARTPOSITION
280 P1=3*SP+1:P2=2*SP
290 POKE650,128:REM TASTENWIEDERHOLUNG
300 PRINT"";:PRINTTAB(SP/2-11)HD$:PRINT""
310 PRINTTAB(SP/2-6)"NEUEINGABE":PRINT
320 PRINTTAB(SP/2-6)"LADEN"
330 GOSUB110:IFA$="L"THEN680
340 IFA$<>"N"THEN330
350 REM ******** NEUEINGABE ************
360 PRINT"DIE EINGABE DER FILES ":PRINT
370 PRINT"MUSS SUKKSESSIV ERFOL";:IFSP<40THENPRINT:PRINT
380 PRINT"GEN.    @@ = ENDE"
390 RESTORE:READA:ZA=A
400 INPUT"FILE-NAMEN";N$
410 IFN$<>"@@"THENN$(A)=N$:A=A+1:GOTO400
420 ZE=A:A=ZA:IF(PEEK(WZ)ANDWW)=WWTHEN450
430 PRINT"DRUECKEN SIE DIE":PRINT"<STOP>-TASTE"
440 PRINT"AM RECORDER":WAITWZ,WW,255-WW
450 PRINT"DRUEKEN SIE DIE ";:IFSP<40THENPRINT:PRINT
460 PRINT"F.FWD-TASTE!"
470 PRINT:PRINT"STARTEN SIE DEN RECOR-";:IFSP<40THENPRINT:PRINT:PRINT"
";
```

```
480 PRINT"DER DURCH DRUECKEN"
490 PRINT:PRINT"EINER TASTE UND ";:IFSP<40THENPRINT:PRINT
500 PRINT"STOPPEN SIE IHN AM "
510 PRINT:PRINT"START DES GEWUENSCH-";:IFSP<40THENPRINT:PRINT:PRINT" ";
520 PRINT"TEN FILES AUF GLEICHE":PRINT"WEISE."
530 WAITWZ,WW,WW:GOSUB150:DI=0
540 GOSUB110:GOSUB120:R=TI:REM CC AN, ZEIT MERKEN
550 PRINT"STOPPEN BEI START:"
560 PRINTN$(A)
570 GOSUB110:GOSUB150:DI=TI-R+DI:REM CC AUS, ZEITDIFFERENZ BERECHNEN
580 ZS(A)=DI-10:DI=DI+10:A=A+1
590 IFA<ZETHEN540:REM LETZTER FILE-NAME ?
600 REM * ABSPEICHERN IN DATA ZEILEN **
610 PRINT"2000DATA"A:REM ANZAHL IN DATAZEILE
620 FORI=ZATOZE-1:PRINT2100+I"DATA"N$(I)","ZS(I)
630 NEXT:PRINT"GOTO650"
640 POKE198,11:FORI=0TO10:POKE631+I,13:NEXT:PRINT:END
650 PRINT"SPULEN SIE JETZT ZU-":PRINT"RUECK UND SPEICHERN":PRINT"DAS
PROGRAMM AM ANFANG"
660 PRINT:PRINT"AB":GOSUB120:END
670 REM ******* LADEN DER FILES *******
680 RESTORE:READA:FORI=0TOA-1:READN$(I),ZS(I):NEXT
690 PRINT"";:PRINTTAB(SP/2-11)HD$
700 PRINT"  F1 = AUFW.":PRINT"  F2 = ABW.":PRINT"  F7 = LADEN"
710 PRINT"":PRINTTAB(SP/2-6)"  S P A C E  ":GOSUB110:ZA=0
720 PRINT"";:PRINTTAB(SP/2-11)HD$:Z=0:PRINT
730 FORI=ZATOZA+8:IFN$(I)<>""THENPRINT:PRINTTAB(3)N$(I)
740 NEXT
750 POKEBS+P1+Z*P2,62:POKEBF+P1+Z*P2,FW
760 GOSUB110:REM TASTATUR ABFRAGE
770 IF(ASC(A$)=133)AND(Z>0)THENPOKEBS+P1+Z*P2,32:Z=Z-1:GOTO750
780 IF(ASC(A$)=134)AND(Z<8)AND(N$(ZA+Z+1)<>"")THENPOKEBS+
P1+Z*P2,32:Z=Z+1:GOTO750
790 IFASC(A$)=136THEN830
800 IFASC(A$)=133ANDZ=0ANDZATHENZA=ZA-9:GOTO720
810 IF(ASC(A$)=134)AND(N$(I))<>""AND(Z=8)THENZA=ZA+9:GOTO720
820 GOTO750
830 REM            LADEN
840 IF(PEEK(WZ)ANDWW)=WWTHEN870
850 PRINT"DRUECKEN SIE DIE":PRINT"<STOP>-TASTE"
860 PRINT"AM RECORDER":WAITWZ,WW,255-WW
870 PRINT"";:PRINTTAB(SP/2-11)HD$:PRINT:PRINT
880 PRINT"DRUECKEN SIE DIE ":PRINT" F.FWD "
890 PRINT:PRINT"TASTE"
900 WAITWZ,WW,WW:REM WARTEN AUF TASTE AM RECORDER
910 PRINT" OK "
920 R=TI:R2=R+ZS(Z+ZA)
930 IFR2<TITHEN950
940 GOTO930
950 GOSUB150
960 PRINTTAB(SP/2-11)HD$
970 PRINT:PRINT"DRUECKEN SIE DIE ":PRINT"      PLAY    "
980 PRINT:PRINT"TASTE"
```

```
990 PRINTTAB(SP/2-6)"  S P A C E  ":GOSUB110
1000 PRINT"lO";CHR$(34)(N$(Z+ZA))CHR$(34)
1010 POKE198,1:POKE631,13:POKE37148,252:PRINTEND
1020 REM ****** PROGRAMMANZAHL ********
2000 DATA 4
2005 REM ****** PROGRAMMDATAS *********
```

10.9 Hardware der Datasette

Bis jetzt habe ich nur die Software zur Datasettenhandhabung beschrieben. In diesem Unterkapitel möchte ich auf die Hardware der Datasette selbst eingehen.

10.9.1 Pflege der Datasette

Wie jeder andere Kassettenrekorder braucht die Datasette auch eine gewisse Pflege. Das häufige Auftreten eines ?LOAD ERROR ist meistens die Folge eines verschmutzten, dejustierten oder magnetisierten Kopfes.

1. Reinigung der Köpfe und der Andruckrolle

Hierzu eignen sich sehr gut handelsübliche Wattestäbchen. Sind sie nicht greifbar, können Sie auch ein Streichholz oder ein anderes Holzstäbchen an einer Seite mit Watte umwickeln. Als Reinigungsmittel empfiehlt sich Alkohol, Fleckenbenzin oder Spiritus benutzen. Zur Reinigung selbst öffnen Sie den Kassettenschacht und drücken die <Play>-Taste. Nun können Sie mit den mit Reinigungsmittel getränkten Wattestäbchen die Köpfe (Lese-/Schreibkopf und Löschkopf) reinigen, indem Sie so lange über die Frontseite der Köpfe fahren, bis die Watte keine braune Färbung mehr annimmt, und anschließend mit einem trockenen Wattestäbchen nachreiben.

Es empfiehlt sich, im gleichen "Abwasch" auch die Gummi-Andruckrolle zu reinigen. Nach längerem Betrieb der Datasette bildet sich nämlich ein schlüpfriger Belag aus Bandabrieb. Dadurch wird das Band nicht mehr gleichmäßig am Tonkopf vorbeigeführt. Zur Reinigung legen Sie ein Stück nicht flusenden Stoffs

(Taschentuch oder ähnliches) über den Zeigefinger, befeuchten es mit einem der oben beschriebenen Reinigungsmittel und halten den Finger bei auf <Play> laufender Datasette von rechts gegen die Andruckrolle.

2. Demagnetisierung

Zur Demagnetisierung benutzen Sie einen käuflichen Demagnetisator oder eine Demagnetisierungskassette, wie sie für Audio-Anlagen erhältlich sind. Die Handhabung ist dann genauso, wie sie in der entsprechenden Bedienungsanleitung beschrieben wird.

10.9.2 Wahl und Handhabung der Kassetten

An eine Kassette zur Daten- und Programmspeicherung braucht man nicht die Anforderungen zu stellen wie an eine Audio-Kassette. Sie brauchen also für Ihre Datasette keine Kassetten mit besonders gutem Frequenzgang oder großem Rauschabstand. Nach dem bisher Gesagten wäre die billigste Kassette gerade gut genug.

Der einzig wichtige Punkt ist, daß die Kassette keine "Drop outs" hat. Drop outs sind Stellen auf dem Band, an denen die Beschichtung mit magnetisierbaren Partikeln nicht einwandfrei ist. An diesen Stellen kann das Band nicht ordnungsgemäß magnetisiert werden, sodaß dort gespeicherte Informationen verlorengehen. Diese Gefahr ist leider bei normalen Super-Billigkassetten gegeben. Meiner Meinung nach fahren Sie deshalb am besten mit den einfachen Kassetten irgend eines renommierten Herstellers. Um zu lange Spulzeiten zu vermeiden, empfehle ich Ihnen maximal C60-Kassetten zu verwenden.

Wie Sie wissen, sind am Anfang oder am Ende des Bandes Vorspannbänder angebracht, um die Belastung, die beim Anschlag an das Ende auf das Band einwirkt, abzufangen. Trotz alledem wird das Magnetband am Anfang und am Ende immer noch stärker belastet als an anderen Stellen, was zu einer Dehnung führen kann. Ich empfehle Ihnen deshalb, auf den ersten und

zu speichern. Sonst gilt für Datenkassetten das gleiche wie für Audio- und Video-Kassetten. Setzen Sie sie keinen starken Magnetfeldern aus und lagern Sie sie staubgeschützt. Auch sollten Sie sie nicht zu lange lagern (über viele Monate), ohne sie einmal umzuspulen. Sonst treten die magnetischen Schichten des Bandes möglicherweise miteinader in Wechselwirkung und können sich dadurch verändern. Um Programme gegen äußere Einflüsse oder auch eigenes Fehlverhalten zu schützen, empfiehlt es sich, von allen wichtigen Daten und Programmen eine Kopie anzufertigen, mit der Sie nicht arbeiten und die Sie auf einer schreibgeschützten Kassette ablegen. So können Sie dann, wenn Ihre Arbeitskopie zerstört wird, auf das Mutter-File zurückgreifen.

10.9.3 Arbeitsweise der Datasette

Wie im Unterkapitel 10.6 beschrieben, wird die zu speichernde Information in Form von Pulsen der Datasette übermittelt. Dies geschieht dadurch, daß beim C64 am Port-Bit 3 des Prozessors (Speicherstelle eins) entweder eine Spannung von ca. 5 V = 1 oder 0 V = 0 anliegt. Dieser Anschluß ist direkt mit den Kassetten-Port-Anschlüssen E und 5 verbunden.

Über einige elektronische Stufen, die ich hier nicht näher erläutern will, gelangt das Signal zum Tonkopf. Der Tonkopf ist eine Spule mit Ringkern, die einen kleinen Schlitz auf der dem Band zugewandten Seite hat.

Entsprechend der an den Spulenanschlüssen angelegten Spannung entsteht ein Magnetfeld am Schlitz, das dann die magnetisierbaren Partikel auf dem Band in eine bestimmte Richtung ausrichtet. Wenn Sie nun das Band lesen, wird der ganze Vorgang umgekehrt.

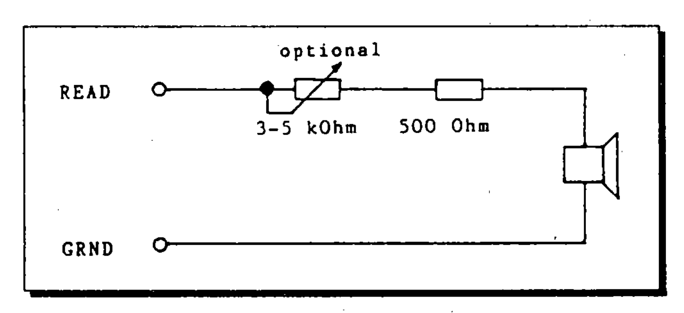
Durch die magnetisierten Partikel wird über den Tonkopfschlitz eine Spannung in der Spule erzeugt, die über einige Baustufen wieder zu dem gleichen Spannungsverlauf führt wie beim Abspeichern. Diese Spannung liegt an den Kassetten-Port-Anschlüssen D und 4 an. Wenn nun aus irgendwelchen Gründen die

Stellung des Tonkopfes zum Band beim Lesen anders ist als beim Speichern, erhalten wir Lesefehler.

10.9.4 Ein Lautsprecher für die Datasette

Sie haben sich bestimmt schon darüber geärgert, daß Sie außer dem ziemlich unzuverlässigen Zählwerk keine Möglichkeit haben, die Position Ihres Bandes festzustellen. Wünschenswert wäre, festzustellen, ob Sie gerade innerhalb eines Programms das nächste suchen oder ob Sie gerade am Start eines Programms sind. So könnten Sie viel schneller ein bestimmtes Programm suchen. Diese Möglichkeit können Sie sich durch den Anschluß eines Lautsprechers an die Datasette schaffen. Mit Hilfe dieses Lautsprechers hören Sie, wo ein Programm anfängt. Sie können nun durch die Wiederholung von kurzzeitigem, schnellem Vorlauf und folgendem "Reinhören" mit <Play> viel besser den Start des folgenden Programms finden.

Weiterhin ist das akustische "feed-back" bei der Tonkopfjustierung, das ich in Unterkapitel 10.9.5. beschreiben werde, ein wichtiges Hilfsmittel. Zum Anschluß an die Datasette eignet sich jeder Lautsprecher. Da es ja nicht auf einen möglichst guten Klanggenuß ankommt, können Sie auch aus einem alten Miniradio den Lautsprecher ausbauen. Dieser Lautsprecher wird an die READ-Leitung angeschlossen. Damit Sie diesen Anschluß nicht überlasten, müssen Sie noch einen Widerstand von ca. 500 Ohm in Reihe schalten. Weiterhin können Sie noch ein Potentiometer von 2 - 5 KOhm in Reihe zum Lautsprecher schalten, um die Lautstärke regeln zu können.



Anschluß des Lautsprechers

Am einfachsten läßt sich ein Lautsprecher an den Stecker der Datasette anschließen. Dazu öffnen Sie den Stecker, der an den Computer gesteckt wird, und klemmen den einen Anschluß an die GROUND-Leitung (A/1) und den anderen an die READ-Leitung (D/4). Schließen Sie den Stecker wieder und stecken ihn an den Kassetten-Port Ihres Computers.

Schalten Sie nun den Computer ein, legen die bespielte Kassette ein und drücken die <Play>-Taste. Jetzt hören Sie entweder einen eintönigen Pfeifton oder ein zirpendes Geräusch. Der Pfeifton entspricht der Synchronisation und kennzeichnet einen Blockanfang. Das zirpende Geräusch entspricht den aufgezeichneten Daten.

10.9.5 Kopfjustage

Es kann aus verschiedenen Gründen vorkommen, daß der Tonkopf nicht mehr in der richtigen Stellung zum Band steht, was zum fehlerhaften Laden führt. Dies kann z.B. passieren, wenn Sie Programme von einer Kassette einlesen wollen, die nicht mit Ihrer Datasette bespielt wurde. Es kommt aber auch vor, daß sich der Kopf selbständig nach längerer Zeit dejustiert. Mit Hilfe eines Lautsprechers ist die Dejustierung gut zu hören. Wenn der Kopf gegenüber dem Band auch nur leicht verwinkelt ist, geht dies meist zu Lasten der hohen Frequenzen.

Doch nun zum praktischen Teil. Wenn Sie die neuere Ausführung der Datasette haben, finden Sie ungefähr in der Mitte vor dem Kassettenfach ein kleines Loch. Durch dieses Loch erreichen Sie mit einem kleinen Kreuzschlitz-Schraubendreher bei gedrückter <Play>-Taste die Tonkopfeinstellschraube.

Doch bevor wir nun beginnen, den Kopf zu justieren, noch zwei Warnungen:

1. Es ist möglich, daß Sie den Kopf so sehr dejustieren, daß eine richtige Justage ohne aufwendige Hilfsmittel (Meßge-

räte) nur sehr schwer möglich ist. Auch müssen Sie darauf achten, daß Sie die Justierschraube nicht gänzlich herausdrehen. Normalerweise reichen maximal zwei Umdrehungen in jede Richtung aus, um den Kopf zu justieren. Meist reicht schon eine halbe Umdrehung aus.

2. Nachdem Sie den Kopf neu justiert haben, kann es sehr gut sein, daß Sie die Programme, die Sie mit der letzten Einstellung abgespeichert haben, nicht mehr laden können. Damit Sie möglichst leicht die ursprüngliche Kopfstellung wiederfinden und einen dejustierten Kopf besser justieren können, benutzen Sie das folgende kleine Programm. Es schreibt ein Zirpen auf Kassette mit einem großen Anteil hoher Frequenzen.

Da es ja die hohen Frequenzen sind, die zuerst gedämpft werden, erleichtern Sie sich durch dieses Zirpen die Einstellung auf die Kopfstellung, mit der das Zirpen abgespeichert wurde, sehr. Auch können Sie damit den Kopf bedeutend genauer justieren. Geben Sie also das Programm ein und speichern es ab. Schreiben Sie dann auf eine leere Kassette ca. zwei Minuten das von diesem Programm erzeugte Zirpen.

```
10 REM KOPFJUSTAGE HILFSPROGRAMM C64
20 E=256*PEEK(56)+PEEK(55)-1:A=E-17
30 FORI=ATOE:READX:POKEI,X:NEXT
40 PRINT"STARTADRESSE"A
50 PRINT:PRINT"LEGEN SIE EINE LEERCASSETTE EIN UND"
60 PRINT:PRINT"DRUECKEN SIE RECORD & PLAY"
70 WAIT1,16,16
80 PRINT:PRINT"ENDE DURCH <RUN/STOP>-TASTE"
90 SYSA:END
1000 DATA160,255,169,255,162,255,32,177,251,136,208,246,32,225,255,208,2
39,96
```

Spulen Sie das Band zurück, nehmen Sie den kleinen Kreuzschlitz-Schraubendreher zur Hand und drücken die <Play>-Taste. Führen Sie nun durch das beschriebene Loch den Schraubendreher senkrecht ein, bis er in der Schraube einrastet. Drehen sie nun jeweils eine halbe Umdrehung nach rechts und links von der Ausgangsposition, und achten Sie dabei auf die typische Klangveränderung. Nur in einer Stellung, nämlich in der Ausgangsstellung, hören Sie die meisten hohen Frequenzen. Legen

Sie nun eine bespielte Datenkassette ein und wiederholen Sie den Vorgang. Sie werden jetzt in einem größeren Bereich keine Klangveränderung mehr feststellen. Die beste Stellung zwischen den hörbaren Änderungen ist hier die Mittelstellung.

Nachdem Sie sich auf diese Weise etwas "eingehört" haben, können Sie auf gleiche Weise den Kopf auch auf Programme justieren, die Sie nicht laden konnten. Versuchen Sie auch hier nach Gehör die beste Kopfstellung herauszufinden. Danach versuchen Sie, das Programm zu laden. Läßt es sich immer noch nicht laden, verändern Sie die Kopfposition etwas und versuchen es erneut.

Gelingt es Ihnen trotz vielfachen Versuchens nicht, das Programm einwandfrei zu laden, verfahren Sie so, wie es in Unterkapitel 10.3.1 beschrieben wurde. Ehe Sie aber das UNNEW-Programm laden, bzw. das Programmfragment abspeichern, müssen Sie den Tonkopf mit Hilfe der von Ihnen angefertigten Justierkassette wieder in die Ausgangsposition bringen.

10.9.6 Andere Kassettenrekorder zur Datenspeicherung

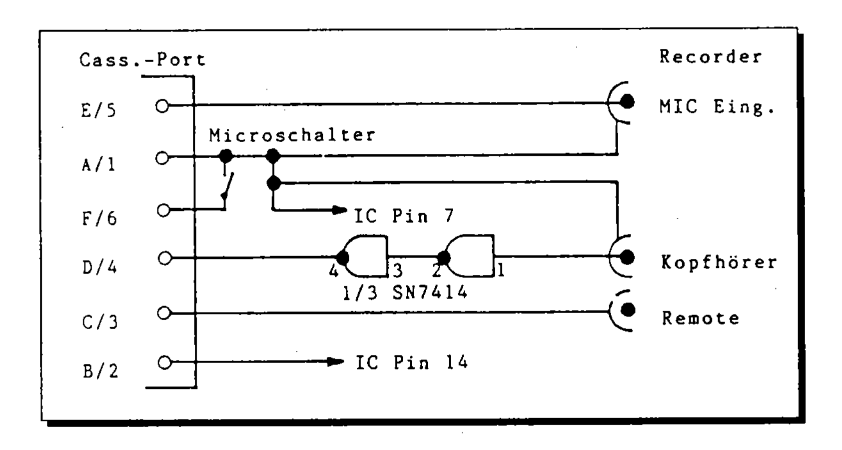
In meinen Ausführungen habe ich mich immer auf die Datasette von Commodore bezogen. Es ist aber ohne weiteres möglich, andere Kassettenrekorder mit einer entsprechenden Anpassung als Aufzeichnungsgerät zu verwenden. Die Anpassung erfolgt durch ein Interface, welches die vom Band kommenden Signale in für den Rechner verständliche Rechteckimpulse umsetzt. Solche Interface-Bausteine werden von verschiedenen Herstellern recht preiswert angeboten.

Der Vollständigkeit halber möchte ich hier eine kleine Schaltung zum Selbstbau darstellen. Die benötigten Bauteile entnehmen Sie bitte folgender Aufstellung.

Bauteile für ein Kassettenrekorder-Interface:

1 Platinen-Stecker für den Kassetten-Port am C64
1 Stecker für den Kopfhörerausgang Ihres Kassettenrekorders
1 Stecker für den Microphoneingang Ihres Kassettenrekorders
1 IC SN 7414 (6 * invertierender Schmitt-Trigger)
1 Microschalter (ein/aus)
1 Stecker für den Remote-Eingang Ihres Rekorders (optional)

Als Rekorder eignen sich für diese Schaltung nur solche, die mit einer Betriebsspannung von 4 bis 6 V arbeiten und den Minuspol auf Masse haben.



Beschreibung der Anschlüsse am Kassetten-Port:

```
A/1 = Masse
B/2 = +5V
C/3 = Kassettenmotor (+5 V = ein)
D/4 = Leseleitung
E/5 = Schalterleitung für Rekordertaste
```

In dem IC SN 7414 sind sechs "invertierende Schmitt-Trigger" enthalten. Schmitt-Trigger sind Schaltungen, die aus einem beliebigen Spannungsverlauf entsprechende Rechteckimpulse bilden, wie ich es im Unterkapitel 10.9.5 beschrieben habe. Von diesen sechs benötigen wir für diese Schaltung nur zwei, da es sich ja um invertierende Schaltungen handelt. Die Betriebsspan-

nung dieses Bausteins entnehmen wir dem Kassetten-Port (Pin7 - A/1, Pin14 - B/2). Wenn Sie den Kassettenrekorder nur für Ihren Rechner benutzen, können Sie die Schaltung in das Batteriefach einbauen, und die Microphon- und Kopfhörerleitung fest anlöten. Damit sparen Sie die Klinkenstecker.

Sehr praktisch wäre es dann noch, wenn Sie in der Rekordermechanik eine Stelle finden würden, wo Sie den Microschalter so anbringen können, daß er beim Drücken einer Taste geschlossen wird. Geht das nicht, müssen Sie den Schalter außen anbringen und manuell betätigen, bevor Sie eine Rekordertaste drücken.

10.10 Ein neues Kassettenbetriebssystem - FastTape

Im ersten Teil habe ich Ihen die Handhabung der Datasette mit dem im Rechner eingebauten Betriebssystem erklärt. Dazu habe ich Ihnen einige Programme gezeigt, die das Arbeiten mit Ihrem Datenrekorder vereinfachen. Nun möchte ich Ihnen ein neues Betriebssystem vorstellen, das nicht nur die schon beschriebenen Vereinfachungen beinhaltet, sondern ca. 10-20 mal schneller arbeitet und damit eine Diskettenstation noch überflügelt.

Ich muß zugeben, daß Sie sich aufgrund der Programmlänge die Finger fast wund tippen können, aber das Ergebnis lohnt die Arbeit. Das Programm hat folgende Charakteristika:

1. 10 mal schnelleres Speichern und Laden der Programme.
2. Unterstützung der Datenspeicherung.

   Daten können bis zu 20 mal schneller geladen und gespeichert werden als mit dem normalen Betriebssystem.
3. "Append" von BASIC-Programmen ist mit einem Befehl möglich.
4. Das Laden in oder das Speichern von bestimmten Speicherbereichen ist ebenfalls mit z7einemz8 Befehl möglich.
5. Programme und Daten, die mit "FastTape" abgespeichert wurden, können vom C64 gelesen werden.

6. Alle FastTape-Befehle arbeiten sowohl im Direktmodus als auch innerhalb von Programmen.

Die Bedienung des Programms

Nach dem Laden des Programms initialisieren sie es mit SYS12*4096. Durch den BASIC-Lader wird das Programm automatisch initialisiert. Nach einem RESET müssen Sie das Programm aber neu initialisieren. Jedem FastTape-Befehl geht ein Pfeil nach links (<-) voraus.

Ihnen stehen jetzt folgende Befehle zur Verfügung (in Klammern stehende Ausdrücke sind optional, müssen also nicht gegeben werden):

SA	**Sekundäradresse**

Sadr, Eadr = Start- bzw. Endadresse

"name" kann auch als String-Variable übergeben werden

S("name",SA,Sadr,Eadr)	**FastTape SAVE**

("name") BASIC-Programm wird als verschiebbares File im Programm gespeichert und mit L an den BASIC-RAM-Start geladen.

S"name",1 BASIC-Programm wird als absolutes File gespeichert und mit L an die Adresse geladen, von wo es abgespeichert wurde.

S"Name",1,Sadr,Eadr
Der Speicherbereich zwischen "Sadr" und "Eadr" wird als absolutes File gespeichert.

L("name",SA,Sadr)	FastTape LOAD

Wird dieser Befehl innerhalb eines Programms gegeben, startet danach das BASIC-Programm analog zum LOAD-Befehl wieder bei der ersten Zeile.

L("name") Ein Programm oder Speicherbereich, der mit S abgespeichert wurde, wird entsprechend der bei S gegebenen SA entweder absolut oder verschiebbar geladen. Wird dieser Befehl im Direktmodus gegeben, werden die BASIC-Vektoren entsprechend dem geladenen Programm gesetzt.

L"name",1 Programme und Speicherbereiche werden absolut geladen. Auch im Direktmodus werden durch diesen Befehl die BASIC-Vektoren "z7nichtz8" geändert. Dadurch ist kein NEW-Befehl nach dem Laden eines Maschinenprogramms notwendig.

L"name,1,Sadr

Ein Speicherbereich wird an "Sadr" geladen, unabhängig davon, von welcher Adresse es abgespeichert wurde. Dadurch haben Sie z.B. die Möglichkeit, einen abgespeicherten Bildschirminhalt auch dann wieder in den Video-ROM einzulesen, wenn Sie den Video-ROM verschoben haben.

V("name",SA,Sadr)	FastTape VERIFY

Arbeitet analog zu L, vergleicht aber nur den Speicherinhalt mit den auf Band stehenden Bytes.

M("name")	FastTape MERGE

Mit diesem Befehl können Sie ein FastTape-Programm an das im Speicher befindliche anhängen.

DS"name"	DATASAVE

Dieser Befehl speichert alle bis dahin definierten Variablen, Felder und Strings ab. Mit diesem Befehl werden nur die im String-Speicher stehenden Strings abgespeichert. Wenn in Ihrem Programm String-Zuweisungen wie

```
100 A$="TEST"
```

existieren, können Sie die entsprechenden Strings nur mit dem gleichen Programm wieder richtig laden. Abhilfe können Sie durch eine Verknüpfung mit einem Leer-String schaffen. Hier gilt das gleiche, wie im Unterkapitel 10.4.1 beschrieben.

DL"Name"	DATALOAD

Mit diesem Befehl laden Sie die mit DS abgespeicherten Variablen, Felder und Strings. Alle bis zu diesem Befehl definierten Variablen, Felder und Strings werden gelöscht.

Achtung!! DS und DL muß immer in Verbindung mit einem File-Namen verwandt werden. Durch diese Befehle werden die Variablen, Felder und Strings in drei Blöcken abgespeichert bzw. geladen. Der File-Name dient dem Computer zur Identifizierung des jeweils ersten Blocks.

Das BASIC-RAM-Ende muß bei DS und dem entsprechenden DL das gleiche sein. Die String-Tabelle wird als absoluter Block gespeichert und geladen. Wenn Sie das Ende des BASIC-RAM heruntersetzen, nachdem Sie DS gegeben haben, und dann mit DL Daten laden, kann eventuell ein hinter das neue BASIC-RAM-Ende geschriebene Maschinenprogramm zerstört werden. Damit Sie sehen, wie komfortabel Sie mit diesen Befehlen Datenverarbeitung betreiben können, finden Sie im nächsten Unterkapitel, ein ausführlich beschriebenes Dateiverarbeitungsprogramm. Durch den BASIC-Lader FT64 wird das Programm im C64 ab der Speicherstelle $C000 abgelegt.

```
82 .PAG 61,5
90 .OPT P
92 :
94 ; FASTTAPE FUER C64
95 : :
98 : :
100 *= $C000
110 COMPU  = $00
120 PORT = $1
130 ABSFLG = $2
140 MOFLAG = $C0
150 BITC = $A3
153 BEFANZ = 4
155 CODE = "_"
157 CHRGET = $73
160 VECTOR = $308 ; BEFEHLSADR. HOLEN
165 IRQFLG = $2A0 ; IRQ FLAG
170 STATUS = $90 ; STATUS-BYTE
175 STARTV = $C3 ; LADESTARTVEKTOR
180 ENDVEC = $AE ; LADEENDVEKTOR
185 PSUMME = $D7 ; PRUEFSUMMEN-BYTE
186 CASPUF = $033C ; STARTADR. VOM CASSETTENPUFFER
187 :CIA1 = $DD00 ; C-64 = STARTADRESSE DES CIA1
188 CIA2 = $DC00 ; C-64 = STARTADRESSE DES CIA2
189 :
190 SAVING = $F68F ; GIBT SAVING 'NAME' AUS
195 LOADING = $F5D2 ; GIBT LOADING AUS
200 STOP = $A82C ;  TESTET <STOP>-TASTE
220 ENDEN = $FC93 ;  LADEN BEENDEN
230 BEFEND = $E17A ;  RUECKSPRUNG VOM LADEN
240 FOUND = $F750 ;  GIBT FOUND 'NAME' AUS
260 STOEND = $F5A9 ;  ENDADR. NACH X/Y
300 OLD = $A7E7 ;  BEFEHL AUSWERTEN
```

```
310 INTER = $A7AE ;  INTERPRETERSCHLEIFE
320 RPTASTE = $F838 ;  WARTET AUF PLAY-TASTE
330 PTASTE = $F817 ;  WARTET AUF  TASTE
335 GABCOL = $B526 ;  GARBAGE COLLECTION
340 WZEICH = $E206 ;  WEITERE ZEICHEN PRINT
350 HFNAM = $E257 ;  FILE-NAMEN HOLEN
360 HOLPAR = $E200 ;  FILE-PARAMETER HOLEN
365 CHRIN = $E20E ;  ZEICHEN HOLEN
370 SYNTAX = $AF08 ;  SYNTAX ERROR AUSGEBEN
380 FRMNUM = $AD8A ;  TERM AUSWERTEN
390 FACINT = $B7F7 ;  IN INTEGER WANDELN
395 CLEAR = $A660 ;  CLR
400 FEHLAUS = $A437 ;  FEHLER AUSGEBEN
410 BEENDEN = $A677 ;
420 :
430 HOLSTA = $FFB7 ; STATUS HOLEN
440 SETNAM = $FFBD ; FILE-NAME-PARAMETER SETZEN
450 SETLFS = $FFBA ; FILE-PARAMETER SETZEN
460 GET = $FFE4 ; EIN ZEICHEN EINLESEN
500 :
510 :
520 ;PROGAMM INITIALISIEREN
530 :
560 INIT LDA #<START
570 STA VECTOR
580 LDA #>START
590 STA VECTOR+1
600 RTS
602 :
603 :
604 ; PROGRAMMSTART
606 :
610 START JSR CHRGET ; ZEICHEN HOLEN
620 BEQ ENDE
630 CMP #CODE ; MIT FASTTAPECODE
640 BEQ FTAPE ; VERGLEICHEN
650 ENDE JMP OLD ; UNGLEICH => ALTE ROUTINE
654 :
655 ;FASTTAPE BEFEHL AUSWERTEN
657 :
660 FTAPE JSR CHRGET
670 JSR SUCH
680 JMP INTER
690 SUCH LDX #0
700 SUCHL CMP TAB,X ; MIT BEFEHLSBUCHSTABEN
710 BEQ BEFEHL ; AUS TABELLE VERGLEICHEN
720 INX
730 CPX BEFANZ
740 BNE SUCHL
750 JMP SYNTAX ; KEIN FASTTAPE BEFEHL
755 :
760 BEFEHL TXA
770 ASL : TAX ; BEFZAHL * 2
```

```
780 LDA BEFADR+1,X ; BEFEHLSADRESSE AUF
790 PHA ; STACK
800 LDA BEFADR,X : PHA
810 JMP CHRGET ; UND BEFEHL AUSFUEHREN
815 :
820 DATA CMP #"S" ; DATEN SPEICHERN
830 BEQ DS
840 CMP #"L" ; DATEN LADEN
850 BEQ DL
860 JMP SYNTAX
865 :
870 DS JSR CHRGET : JMP  DASAV1
880 DL JSR CHRGET : JMP DATLOD
885 :
886 ;TABELLE DER BEFEHLSBUCHSTABEN
887 :
890 TAB .ASC "SLVMD"
895 :
896 ;TABELLE DER BEFEHLSADRESSEN
897 :
900 BEFADR .WORD SAVE-1 : .WORD LOAD-1 : .WORD VERIFY-1
910 .WORD MERGE1-1 : .WORD DATA-1
920 :
1650 :
1660 :
1665 ;SAVE ROUTINE
1666 :
1670 SAVE LDX #$05 ; ANZAL DER SYNCHR.
1680 STX $AB ; WIEDERHOLUNGEN
1682 LDX #$00 ; FLAG LOESCHEN
1684 STX ABSFLG
1690 JSR GETPARA ; PARAMETER HOLEN
1692 LDA ABSFLG
1694 AND #2 ; TEST AUF BIT 2
1695 BNE ABSOLUT ; GESETZT => ABSOLUT LADEN
1700 LDX #$04
1710 LOOP1 LDA $2A,X ; SPEICHERT BASICSTART UM
1720 STA $AB,X
1725 STA $A6,X
1730 DEX
1740 BNE LOOP1
1750 ABSOLUT JSR RPTASTE
1760 JSR SAVING ; 'SAVING NAME' AUSGEBEN
1770 JSR MOTOR ; MOTOR EINSCHALTEN
1774 :
1775 ;VORSPANN SCHREIBEN
1776 :
1780 JSR SYNCH ; SYBCHRONISATION 1 SCHREIBEN
1790 LDA $B9 ; SECUNDAERADR. INCREMENTIEREN
1800 CLC
1810 ADC #$01
1820 DEX
1830 JSR WBYTE ; SCHREIBT SA
```

```
1840 LDX #$08+COMPU
1850 LOOP2 LDA $A7,Y ; SCHREIBT START- UND ENDADRESSE
1860 JSR WBYTE
1870 LDX #$06+COMPU
1880 INY
1890 CPY #$05
1900 NOP
1910 BNE LOOP2
1920 LDY #$00 ;SCHREIBT FILE-NAMEN UND 'SPACES'
1930 LDX #$04+COMPU
1940 LOOP3 LDA ($BB),Y
1950 CPY $B7
1960 BCC TEXT
1970 LDA #$20
1980 DEX
1990 TEXT JSR WBYTE
2000 LDX #$05
2010 INY
2020 CPY #$BB
2030 BNE LOOP3
2034 :
2035 ;SYNCHRONISATION 2 SCHREIBEN
2036 :
2040 LDA #$02
2050 STA $AB
2060 JSR SYNCH ; SCHREIBT SYNCHRONISATION
2070 TYA
2080 JSR WBYTE ; 0 BYTE => ENDE VORSPANN
2090 STY PSUMME ; PRUEFSUMMEN-BYTE LOESCHEN
2100 LDX #$07+COMPU
2104 :
2105 ;PROGRAMM SCHREIBEN
2106 :
2110 NOP
2120 PRGLOOP LDA ($AC),Y ; PROGRAMM AUF BAND SCHREIBEN
2130 JSR WBYTE
2140 LDX #$03+COMPU
2150 INC $AC ; PROGRAMMZEIGER ERHOEHEN
2160 BNE NOHI
2170 INC $AD
2180 DEX
2190 DEX
2200 NOHI LDA $AC
2210 CMP ENDVEC ; PROGRAMMENDE EREICHT
2220 LDA $AD
2230 SBC ENDVEC+1
2240 BCC PRGLOOP ; NEIN => WEITER
2250 NOP
2260 LOOP4 LDA PSUMME ; PRUEFSUMME SCHREIBEN
2270 JSR WBYTE
2280 LDX #$07+COMPU
2290 DEY
2300 BNE LOOP4
```

```
2310 INY
2320 STY MOFLAG ; MOTOR AUSSCHALTEN VORBEREITEN
2330 CLI
2340 CLC
2350 LDA #$00
2360 STA IRQFLG
2370 JMP ENDEN
2372 :
2373 ;MOTOR STARTEN
2374 :
2375 MOTOR JSR STOP
2380 LDY #$00
2390 STY MOFLAG
2400 LDA $D011 ; BILDSCHIRM AUS
2410 AND #$EF ;
2420 STA $D011 ;
2430 ANLOOP DEX  ; WARTET BIS MOTOR HOCH
2440 BNE ANLOOP ; GELAUFEN IST
2450 DEY
2460 BNE ANLOOP
2470 SEI
2480 RTS
2481 :
2482 ; SYNCHRONISATION GENERIEREN
2484 :
2490 SYNCH LDY #$00
2500 LOOP5 LDA #$02 ; START-BYTE 255-9 MAL SCHREIBEN
2510 JSR WBYTE
2520 LDX #$07+COMPU ; BYTE-WERT
2530 DEY
2540 CPY #$09
2550 BNE LOOP5
2560 LDX #$05+COMPU
2570 DEC $AB
2580 BNE LOOP5
2581 :
2582 ; COUNTDOWN SCHREIBEN
2584 :
2590 COUNTS TYA ; COUNTDOWN
2600 JSR WBYTE
2610 LDX #$07+COMPU
2620 DEY
2630 BNE COUNTS
2640 DEX
2650 DEX
2660 RTS
2662 :
2664 ;BYTE AUF BAND SCHREIBEN
2666 :
2670 WBYTE STA $BD
2680 EOR PSUMME
2690 STA PSUMME
2700 LDA #$08
```

```
2710 STA BITC
2720 SHIFT ASL $BD
2730 LDA PORT
2740 AND #$F7
2750 JSR T1LOOP
2760 LDX #$11+COMPU
2770 NOP
2780 ORA #$08
2790 JSR T1LOOP
2800 LDX #$0E+COMPU
2810 DEC BITC
2820 BNE SHIFT
2830 RTS
2840 T1LOOP DEX
2850 BNE T1LOOP
2860 BCC BIT0 ; WENN CARRY GESETZT,
2870 LDX #$0B ; DANN ZEITSCHLEIFE
2880 T2LOOP DEX ; VERLAENGERN
2890 BNE T2LOOP
2900 BIT0 STA PORT
2902 RTS
2903 :
2904 :
2905 ;MERGE
2906 :
2909 ;MERGE
2910 MERGE1 LDA $2D
2911 SEC
2912 SBC #2 : TAY : LDA $2E :SBC #0
2913 LDX #0
2914 BEQ MERGE2
2915 :
2916 :
2917 ;LADEN
2918 :
2919 :
2920 LOAD LDX #$00
2925 .BYTE $2C
2930 VERIFY LDX #$01
2940 LDY $2B ; BASICSTARTVECTOREN IN
2950 LDA $2C ; STARTSPEICHERSTELLEN
2960 MERGE2 STX $0A ; LOAD FLAG (0=LOAD/1=VERIFY)
2970 STX $93 ; SETZEN
2975 LDX #0
2980 STX ABSFLG ; ABSOLUTFAG LOESCHEN
2985 STY STARTV
2990 STA STARTV+1
3000 JSR GETPARA
3010 JSR LOADR ; LOAD UND VERIFY
3020 JMP BEFEND
3022 :
3024 ;LADEROUTINE
3026 :
```

```
3040 LOADR JSR SSYNCH
3050 LDA $AB
3060 CMP #$02 ; IST PRG ABS. GESPEICHERT
3070 BEQ ABSOL ; => ABSOLUT LADEN
3080 CMP #$01 ; BYTE NICHT 1 => WEITERSUCHEN
3090 BNE LOADR
3100 LDA $B9 ; SEKUNDAERADRESSE = 0
3110 BEQ RELLOD ; => VERSCHIEBLICH LADEN
3112 ABSOL LDA #2 ; ABSOLUT-FLAG SETZEN
3113 ORA ABSFLG
3114 STA ABSFLG
3120 LDA CASPUF ; STARTADRESSE AUS PUFFER
3130 STA STARTV ; IN STARTVECTOR
3140 LDA CASPUF+1
3150 STA STARTV+1
3155 RELLOD JSR ADTEST
3156 LDA ABSFLG ; TEST AUF 'WARTEN'
3157 AND #4
3158 BNE NOWAIT
3160 JSR FOUND ; FOUND 'NAME' AUSGEBEN
3170 WAIT JSR GET ;
3180 BEQ WAIT ;
3190 NOWAIT JSR STOP
3200 LDY $B7 ; FILE-NAMEN-LAENGE
3210 BEQ NONAME
3220 TESTNA DEY ; FILE-NAMEN TESTEN
3230 LDA ($BB),Y
3240 CMP CASPUF+5,Y
3250 BNE LOADR
3260 TYA
3270 BNE TESTNA
3280 NONAME STY STATUS ; STATUS LOESCHEN
3290 JSR LOADING
3300 LDA CASPUF+2 ; ENDADRESSE BERECHNEN
3310 SEC ; AUS DIFFERENZ START- + ENDADR.
3320 SBC CASPUF ; AUS DEM HEADER
3330 PHP ; PLUS BESTIMMTER
3340 CLC ; STARTADRESSE
3350 ADC STARTV
3360 STA ENDVEC
3370 LDA CASPUF+3
3380 ADC STARTV+1
3390 PLP
3400 SBC CASPUF+1
3410 STA ENDVEC+1
3420 JSR PLOAD
3430 LDA $BD
3440 EOR PSUMME ; PRUEFSUMME TESTEN
3450 ORA STATUS
3460 BEQ OK
3470 LDA #$FF
3475 STA STATUS
3480 OK LDA ABSFLG ; FLAG TESTEN
```

```
3484 BNE ALTADR ; WENN GESETZT=> ALTE ENDVECTORE
3486 JMP STOEND ; ERHALTEN
3488 ALTADR LDX $2D
3490 LDY $2E
3492 RTS
3500 SSYNCH JSR SCOUNT ; SYNCHRONISATION SUCHEN
3510 CMP #$00 ; SYNCHR. VOR HEADER =>
3520 BEQ SSYNCH ; SYNCH NACH HEADER SUCHEN
3530 STA $AB ; SEKUNDAERADRESSE + 1
3540 SPSTART JSR HOLBYT ; START DES PROGRAMMS
3550 STA ($B2),Y ; SUCHEN
3560 INY
3570 CPY #$C0
3580 BNE SPSTART
3590 BEQ GEFUN ; ENDE VORSPANN,PRG LADEN
3600 PLOAD JSR SCOUNT ; PROGRAMM LADEN
3610 LLOOP JSR HOLBYT
3620 CPY $93
3630 BNE VERGL ; BEI VERIFY NUR VERGLEICHEN
3640 STA (STARTV),Y
3650 VERGL CMP (STARTV),Y
3660 BEQ GLEICH
3670 STX STATUS
3680 GLEICH EOR PSUMME ; PRUEFSUMME BERECHNEN
3690 STA PSUMME
3700 INC STARTV ; ADRESSE ERHOEHEN
3710 BNE NOTHI
3720 INC STARTV+1
3730 NOTHI LDA STARTV
3740 CMP ENDVEC ; ENDADRESSE ERREICHT
3750 LDA STARTV+1
3760 SBC ENDVEC+1
3770 BCC LLOOP ; NEIN => WEITER
3780 JSR HOLBYT
3790 JSR MOTOR
3800 INY
3810 GEFUN STY MOFLAG
3820 CLI
3830 CLC
3840 LDA #$00
3850 STA IRQFLG
3860 JMP ENDEN
3870 SCOUNT JSR PTASTE ; COUNTDOWN SUCHEN
3880 JSR MOTOR
3890 STY PSUMME
3900 LDA #$07 ; ($27) WERT FUER TIMER LO
3910 STA CIA1+6 ; (PORT+8) IN TIMER LO
3920 LDX #$01 ; WERT FUER TIMER HI
3930 SSTART JSR HOLBIT ; START SUCHEN
3940 ROL $BD
3950 LDA $BD
3960 CMP #$02 ; START GEFUNDEN
3970 BNE SSTART
```

```
3980 LDY #$09 ; => COUNTDOWN SUCHEN
3990 ENDE2 JSR HOLBYT ; ENDE DER '2' - BYTES
4000 CMP #$02 ; SUCHEN
4010 BEQ ENDE2
4020 COUNTL CPY $BD ; TESTET DEN COUNTDOWN
4030 BNE SSTART
4040 JSR HOLBYT
4050 DEY
4060 BNE COUNTL
4070 RTS
4072 :
4075 ; BYTE VON BAND HOLEN
4076 :
4080 HOLBYT LDA #$08 ; 8 BIT
4090 STA BITC
4100 SHIFT7 JSR HOLBIT
4110 ROL $BD ; SCHIBT CARRY IN EINGABEPUFFER
4120 NOP
4130 NOP
4140 NOP
4150 DEC BITC
4160 BNE SHIFT7
4170 LDA $BD
4180 RTS
4182 :
4184 ; BIT VON BAND HOLEN FUER C-64
4185 :
4190 HOLBIT LDA #$10
4200 WAITDA BIT CIA2+13 ;WARTET AUF SIGNAL AN 'FLAG'
4210 BEQ WAITDA
4220 LDA CIA1+13 ; ICR LADEN. HAT TB 0 ERREICHT
4225 ; => BIT 1 IST GESETZT
4230 STX CIA1+7 ; WERT IN TB HI
4240 PHA ; ICR RETTEN
4250 LDA #$19 ; TB STARTEN, EINMALIGES ZAEHLEN
4260 STA CIA1+15
4270 PLA ; ICR HOLEN
4280 LSR A ; BIT 1 INS CARRY SCHIEBEN
4290 LSR A
4300 RTS
4310 :
4435 ;PARAMETER HOLEN
4436 :
4450 GETPARA LDA #0
4460 JSR SETNAM
4470 LDX #1 ; DEFAULT FUER GA
4480 LDY #00 ; DEFAULT FUER SA
4490 JSR SETLFS
4510 JSR WZEICH ; WEITERE ZEICHEN
4520 JSR HFNAM
4540 JSR WZEICH
4550 JSR HOLPAR
4560 TXA
```

```
4570 TAY
4580 JSR SETLFS
4600 JSR WZEICH
4610 JSR HOLWERT
4620 STX $AC
4622 STX $A7
4624 STY $A8
4630 STY $AD
4632 LDX #1 ; FLAG FUER AN BESTIMMTE
4634 STX ABSFLG ; ADRESSE LADEN UND RETTEN
4650 JSR WZEICH
4660 JSR HOLWERT
4670 STX $AE
4672 STX $A9
4674 STY $AA
4680 STY $AF
4685 LDA ABSFLG
4690 ORA #2
4695 STA ABSFLG
4700 RTS
4710 HOLWERT JSR CHRIN ; NAECHSTEN WERT HOLEN
4720 JSR FRMNUM ; AUSDRUCK AUSWERTEN
4730 JSR FACINT ; IN 2BYTE WERT UMRECHNEN
4740 LDX $14
4750 LDY $15
4760 RTS
4770 ADTEST LDA ABSFLG ; FLAG LADEN
4780 AND #1
4790 BEQ NORM ; UNGLEICH => NORMAL LADEN
4800 LDA $AC
4810 STA STARTV
4820 LDA $AD
4830 STA STARTV+1
4840 NORM RTS
4940 :
4945 :
4950 ; DATASAVE
4970 :
5000 DASAV1 JSR GABCOL
5010 LDX #5
5020 STX $AB
5030 JSR GETPARA
5040 LDA #1  ; SA=1
5050 STA $B9
5060 LDX #4
5070 ULOOP LDA $2C,X ; VARIABLEN START- UND
5080 STA $AB,X ; ENDADRESSEN UMSPEICHERN
5085 STA $A6,X
5090 DEX
5100 BNE ULOOP
5110 JSR ABSOLUT ; ABSPEICHERN
5112 LDX #0
5114 STX $B7 ; KEIN FILE-NAME
```

```
5115 LDX #5
5117 STX $AB
5119 DEX
5130 U2LOOP LDA $2E,X ; ARRAY START- UND
5140 STA $AB,X ; ENDADRESSEN UMSPEICHERN
5145 STA $A6,X ; ENDADRESSEN UMSPEICHERN
5150 DEX
5160 BNE U2LOOP
5170 JSR ABSOLUT ; ABSPEICHERN
5175 LDX #5
5177 STX $AB
5190 LDA $33 ; STRING START- UND
5200 LDX $34 ;
5210 STA $AC
5215 STA $A7
5220 STX $AD
5225 STX $A8
5230 LDA $37 ; ENDADRESSEN UMSPEICHERN
5240 LDX $38 ;
5250 STA $AE
5255 STA $A9
5260 STX $AF
5265 STX $AA
5270 JMP ABSOLUT ; ABSPEICHERN
5300 :
5305 :
5310 ; DATALOAD
5320 :
5410 DATLOD JSR GETPARA
5420 LDA #1 ; AN BESTIMMTE ADRESSE LADEN
5430 STA ABSFLG
5440 LDA $2D ; VARIABLENSTARTADRESSE ALS
5450 LDX $2E ; STARTADRESSE
5460 STA $AC ; UMSPEICHERN
5470 STX $AD
5480 JSR LOADR ; LADEN
5482 LDA #4+1 ; NICHT WARTEN, AN BESTIMMTE
5486 STA ABSFLG ; ADRESSE LADEN
5490 LDA #0
5500 STA $B7 ; KEIN FILE-NAME
5510 LDX $AE ; VARIBLENENDE
5520 LDY $AF
5530 STX $2F ; IN VEKTOR
5540 STY $30
5550 STX $AC ; UND LADESTARTVEKTOR
5560 STY $AD
5570 JSR LOADR ; LADEN
5580 LDX $AE ; ARRAYENDE
5590 LDY $AF
5600 STX $31 ; IN VEKTOR
5610 STY $32
5620 LDA #4 ; VON ADRESSE VOM HEADER
5630 STA ABSFLG ; LADEN
```

```
5640 JSR LOADR ; LADEN
5642 LDA CASPUF : STA $33
5644 LDA CASPUF+1 : STA $34
5650 JSR HOLSTA ; STATUS HOLEN
5660 AND $BF ; EOF-BIT LOESCHEN
5670 BEQ NOFEHL ; KEIN FEHLER
5680 LDX #$1D ; OFFSET FUER 'LOAD ERROR'
5690 JMP FEHLAUS ; FEHLER AUSGEBEN
5700 NOFEHL LDX #$19 ; STRING-DESCRIPTOR
5710 STX $16 ; INDEX RUECKSETZEN
5720 LDA #00
5730 STA $3A : STA $10 ;CONT SPERREN
5740 RTS
5750 .END
```

```
1000 REM LOADER FT64
1010 E=50105:A=49152:PS=0
1020 FORI=ATOE:READX:POKEI,X:PS=PS+X:NEXT
1030 IFPS<>120504THENPRINT"FEHLER IN DATAS":END
1040 SYSA
10000 DATA169,11,141,8,3,169,192,141,9,3,96,32,115,0,240,4,201,95,240,3,
76,231
10010 DATA167,32,115,0,32,32,192,76,174,167,162,0,221,84,192,240,8,232,2
28,4
10020 DATA208,246,76,8,175,138,10,170,189,90,192,72,189,89,192,72,76,115
,0,201
10030 DATA83,240,7,201,76,240,9,76,8,175,32,115,0,76,17,195,32,115,0,76,
97,195
10040 DATA83,76,86,77,68,98,192,111,193,114,193,97,193,60,192,162,5,134,
171,162
10050 DATA0,134,2,32,176,194,165,2,41,2,208,11,162,4,181,42,149,171,149,
166,202
10060 DATA208,247,32,56,248,32,143,246,32,252,192,32,19,193,165,185,24,1
05,1
10070 DATA202,32,51,193,162,8,185,167,0,32,51,193,162,6,200,192,5,234,20
8,242
10080 DATA160,0,162,4,177,187,196,183,144,3,169,32,202,32,51,193,162,5,2
00,192
10090 DATA187,208,237,169,2,133,171,32,19,193,152,32,51,193,132,215,162,
7,234
10100 DATA177,172,32,51,193,162,3,230,172,208,4,230,173,202,202,165,172,
197,174
10110 DATA165,173,229,175,144,231,234,165,215,32,51,193,162,7,136,208,24
6,200
10120 DATA132,192,88,24,169,0,141,160,2,76,147,252,32,44,168,160,0,132,1
92,173
10130 DATA17,208,41,239,141,17,208,202,208,253,136,208,250,120,96,160,0,
169,2
10140 DATA32,51,193,162,7,136,192,9,208,244,162,5,198,171,208,238,152,32
,51,193
10150 DATA162,7,136,208,247,202,202,96,133,189,69,215,133,215,169,8,133,
163,6
```

```
10160 DATA189,165,1,41,247,32,85,193,162,17,234,9,8,32,85,193,162,14,198
,163
10170 DATA208,233,96,202,208,253,144,5,162,11,202,208,253,133,1,96,165,4
5,56
10180 DATA233,2,168,165,46,233,0,162,0,240,9,162,0,44,162,1,164,43,165,4
4,134
10190 DATA10,134,147,162,0,134,2,132,195,133,196,32,176,194,32,142,193,7
6,122
10200 DATA225,32,10,194,165,171,201,2,240,8,201,1,208,243,165,185,240,16
,169
10210 DATA2,5,2,133,2,173,60,3,133,195,173,61,3,133,196,32,2,195,165,2,4
1,4,208
10220 DATA8,32,80,247,32,228,255,240,251,32,44,168,164,183,240,11,136,17
7,187
10230 DATA217,65,3,208,191,152,208,245,132,144,32,210,245,173,62,3,56,23
7,60
10240 DATA3,8,24,101,195,133,174,173,63,3,101,196,40,237,61,3,133,175,32
,31,194
10250 DATA165,189,69,215,5,144,240,4,169,255,133,144,165,2,208,3,76,169,
245,166
10260 DATA45,164,46,96,32,88,194,201,0,240,249,133,171,32,134,194,145,17
8,200
10270 DATA192,192,208,246,240,45,32,88,194,32,134,194,196,147,208,2,145,
195,209
10280 DATA195,240,2,134,144,69,215,133,215,230,195,208,2,230,196,165,195
,197
10290 DATA174,165,196,229,175,144,221,32,134,194,32,252,192,200,132,192,
88,24
10300 DATA169,0,141,160,2,76,147,252,32,23,248,32,252,192,132,215,169,7,
141,6
10310 DATA221,162,1,32,153,194,38,189,165,189,201,2,208,245,160,9,32,134
,194
10320 DATA201,2,240,249,196,189,208,232,32,134,194,136,208,246,96,169,8,
133,163
10330 DATA32,153,194,38,189,234,234,234,198,163,208,244,165,189,96,169,1
6,44
10340 DATA13,220,240,251,173,13,221,142,7,221,72,169,25,141,15,221,104,7
4,74
10350 DATA96,169,0,32,189,255,162,1,160,0,32,186,255,32,6,226,32,87,226,
32,6
10360 DATA226,32,0,226,138,168,32,186,255,32,6,226,32,244,194,134,172,13
4,167
10370 DATA132,168,132,173,162,1,134,2,32,6,226,32,244,194,134,174,134,16
9,132
10380 DATA170,132,175,165,2,9,2,133,2,96,32,14,226,32,138,173,32,247,183
,166
10390 DATA20,164,21,96,165,2,41,1,240,8,165,172,133,195,165,173,133,196,
96,32
10400 DATA38,181,162,5,134,171,32,176,194,169,1,133,185,162,4,181,44,149
,171
10410 DATA149,166,202,208,247,32,127,192,162,0,134,183,162,5,134,171,202
,181
```

```
10420 DATA46,149,171,149,166,202,208,247,32,127,192,162,5,134,171,165,51
,166
10430 DATA52,133,172,133,167,134,173,134,168,165,55,166,56,133,174,133,1
69,134
10440 DATA175,134,170,76,127,192,32,176,194,169,1,133,2,165,45,166,46,13
3,172
10450 DATA134,173,32,142,193,169,5,133,2,169,0,133,183,166,174,164,175,1
34,47
10460 DATA132,48,134,172,132,173,32,142,193,166,174,164,175,134,49,132,5
0,169
10470 DATA4,133,2,32,142,193,173,60,3,133,51,173,61,3,133,52,32,183,255,
37,191
10480 DATA240,5,162,29,76,55,164,162,25,134,22,169,0,133,58,133,16,96
```

10.10.1 Programmbeschreibung

Für interessierte Leser, die zumindest einige Grundkenntnisse in Assembler-Programmierung haben, möchte ich das FastTape-Programm näher beschreiben. Vielleicht fallen Ihnen noch einige Verbesserungen ein, die Sie in das Programm einbringen wollen.

Prinzipielle Arbeitsweise

Einzelne Bits werden durch Rechteckpulse auf Band geschrieben. Das Byte 20 entspricht Binär %00010100 und sieht folgendermaßen aus:

```
Wert    0 0 0 1 0 1 0 0
Bit-Nr. 7 6 5 4 3 2 1 0
```

In folgendem Speicherformat wird ein Programm gespeichert:

1. Synchronisation 1
   5mal (246mal Byte 2). Dadurch wird beim Ladevorgang auf den Start eines Bytes synchronisiert.
   Countdown-Bytes 9, 8, 7, 6, 5, 4, 3, 2, 1. Hierdurch wird das Ende der Synchronisation gekennzeichnet.
2. Vorspann, bestehend aus

```
Sekundäradresse-1
Start- und Endadresse
File-Namen
```

Der Vorspann wird immer auf 192 Zeichen mit <SPACE> aufgefüllt und beim Laden im Kassettenpuffer abgelegt. Somit ist es auch hier möglich, mit dem File-Namen Maschinenprogramme zu übergeben, wie es im Unterkapitel 10.5.3 beschrieben wurde.
3. Synchronisation - Wie Synchronisation 1, aber nur 2mal.
4. Null-Byte-Kennzeichnung dafür, daß jetzt die Daten folgen.
5. Daten
6. Prüfsumme, 255mal

Im Unterprogramm INIT wird der Vektor zur Interpreter-Schleife auf das FastTape-Programm umgelenkt. Dadurch wird erreicht, daß jedesmal, wenn ein Befehl ausgewertet werden soll, FastTape angesprungen wird. Im Unterprogramm START wird nun getestet, ob das erste Zeichen der FastTape-Code (Pfeil nach links) ist. Ist das Zeichen kein FastTape-Code, wird in die Interpreter-Schleife zurückgesprungen.

Wird der FastTape-Code gefunden, wird über das Unterprogramm SUCH durch Vergleich mit der Buchstabentabelle TAB die entsprechende Startadresse bestimmt. Wird der Buchstabe nicht gefunden, wird ein ?SYNTAX ERROR ausgegeben.

Sonst wird die entsprechende Befehlsstartadresse auf den STACK geschoben. Dadurch verzweigt das Programm nach dem RTS der CHRGET-Routine zu der entsprechenden Adresse plus eins und kehrt nach Ausführung zu der Adresse $C01D($701D) zurück. Bevor ich die einzelnen Hauptprogramme erkläre, möchte ich einige häufig benutzte Unterprogramme beschreiben.

Byte auf Band schreiben: WBYTE $C133

Die Übergabe des zu schreibenden Bytes erfolgt über den Akkumulator. Dieses Byte wird in die Speicherstelle $BD geschrieben und dann mit der EXOR-Prüfsumme durch EXKLUSIV ODER verknüpft.

In $C139 wird 8 (ein Byte = acht Bit) in den Bitcounter geschrieben. Mit SHIFT beginnt die Schleife, die acht Bit auf Band schreibt. Zuerst wird das höchstwertige Bit in das Carry-

Flag geschoben. Das PORT-Byte wird nun in den Akku geladen und das Schreib-Byte (Bit drei) gelöscht.

Entsprechend der Zeitschleife T1LOOP (bei gesetzten Carry-Flag zusätzlich T2LOOP) ist die Schreibleitung nun "high". Am Ende der Zeitschleife wird der Akku-Inhalt in den Port übertragen mit der Folge, daß die Schreibleitung "low" wird.

In $C149 wird dann im Akku wieder Bit drei gesetzt und erneut in die Zeitschleife T1LOOP verzweigt. Am Ende wird dann durch die Übertragung des Akku-Inhalts in den Port die Schreibleitung wieder "high".

In $C150 wird der Bitcounter dekrementiert und getestet. Wurden alle acht Bits übertragen, wird zurückgesprungen.

Byte vom Band lesen: HOLBYT $C286

Zuerst wird ab $C286 der Bitcounter auf 8 gesetzt. Dann werden acht Bit nacheinander über HOLBIT eingeladen und über das Carry-Flag in $BD geschoben. Am Ende ($C269) wird über den Akku das eingelesene Byte übergeben.

Bit vom Band lesen: HOLBIT $C299

Die Schleife $C299 - $C29E wartet bis Bit vier im ICR2 des CIA2 "high" wird. Dieses Bit wird gesetzt, wenn an der READ Leitung (D/4) des Kassetten-Ports ein Signal anliegt. Dann wird das ICR1 geladen und das X-Register als High-Byte in den Zähler B vom CIA2 geschrieben und ICR1 auf den STACK gerettet.

In $C2A9 werden Bit 1,3 und 4 im Kontroll-Register B gesetzt, was zur Folge hat, daß Zähler B einmal abwärts zählt und gestartet wird. Danach wird der gerettete ICR1-Inhalt zurückgeholt und durch zweimaliges Rechtsschiften Bit 1 ins Carry-Flag geschoben. Dieses Bit wird dann "high", wenn der Zähler bis auf 0 gezählt hat, bevor er neu gestartet wurde.

Auf diese Weise wird festgestellt, ob es lange dauerte, bis die Leseleitung wieder "high" war, was dem gesetzten Bit entspricht.

Parameter holen: GETPARA $C2B0

Diese Routine holt die mit den Befehlen übergebenen Parameter und schreibt sie in die entsprechenden Speicherstellen. Weiterhin wird entsprechend der Parameter das Flag "ABSFLG" gesetzt. Dieses Flag ist ein Binär-Flag und steuert folgende Funktionen:

Bit	Wert	Wirkung
0	1	1 = File an eine beim L-Befehl übermittelte Adresse laden
1	2	1 = File absolut laden oder speichern
2	4	1 = Beim C-64 nicht warten, sondern sofort laden

Synchronisation schreiben: SYNCH $C113

Dieses Unterprogramm schreibt so oft, wie in $AB angegeben ist, 246mal das Byte 2 auf Band. Zuletzt wird ein Countdown aus den Bytes 9, 8, 7, 6, 5, 4, 3, 2, 1 auf Band geschrieben.

Synchronisation suchen: SCOUNT $C258

Zuerst wird die Rekordertaste abgefragt. Dann wird der Motor gestartet, Bildschirm abgeschaltet und das Prüfsummen-Byte auf 0 gesetzt. Ab $C260 wird 7 in das Low-Byte geschrieben und das X-Register mit für das High-Byte des Timers mit 1 geladen

Ab SSTART werden dann so lange Bits eingelesen, bis der Byte-Wert der eingelesenen Bits genau 2 wird. Dann werden alle folgenden Zweier-Bytes der Synchronisation eingelesen und festgestellt, ob danach ein Countdown ab 9 folgt. Wird ein fehlerfreier Countdown gefunden, wird zum Hauptprogramm zurückgesprungen.

Vorspann einlesen: SSYNCH $C20A

Dieses Unterprogramm sucht mit SCOUNT die nächste Synchronisation. Mit Hilfe des letzten von SCOUNT eingelesenen Bytes stellt es fest, ob es sich um Synchronisation 1 oder 2 handelt. Ist die gefundene Synchronisation nicht Synchronisation 1, wird weitergesucht. Sonst wird das letzte Byte als Sekundäradresse in $AB gerettet. Daraufhin wird der Vorspann eingelesen und im Kassettenpuffer abgelegt. Danach wird über die Betriebssystem-Routine ab $FC93 der Motor ausgeschaltet und der Bildschirm wieder eingeschaltet.

Programm laden: PLOAD $C21F

Durch dieses Unterprogramm wird das Programm eingelesen. Zuerst wird mit SCOUNT die Synchronisation 2 gesucht und damit auch die Leseroutine auf das Band synchronisiert.

LLOOP ist die eigentliche Laderoutine. Mit HOLBYT wird ein Byte eingelesen und über den Startvektor "indirekt" indiziert in den Speicher geschrieben und verglichen. Bei V (VERIFY) wird nur verglichen. Tritt dabei ein Fehler auf, wird das Status-Byte gesetzt. Danach wird die EXOR-Prüfsumme gebildet und der Startvektor inkrementiert.

Die Schleife LLOOP wird so lange durchlaufen, bis die Differenz zwischen Start- und Endvektor gleich null ist. Dann wird noch die abgespeicherte Prüfsumme eingelesen, der Motor aus- und beim C64 der Bildschirm angeschaltet. Über die Betriebssystem-Routine $FC93 wird zum Hauptprogramm zurückgesprungen. Als letztes möchte ich die Hauptroutinen etwas näher beschreiben.

```
SAVE ROUTINE       $C063
```

In $AB wird zunächst die Wiederholungszahl der Synchronisation 1 abgelegt und das Flag ABSFLG gelöscht. Danach werden über GETPARA die einzelnen File-Parameter eingelesen. Entsprechend ABSFLG werden dann die BASIC-Vektoren in die Lade-, Start- und Endevektoren übertragen.

Ab $C07F wird die Rekordertaste abgefragt, "SAVING name" ausgegeben, der Motor gestartet und der Bildschirm ausgeschaltet. Über das Unterprogramm SYNCH wird nun die Synchronisation auf Band geschrieben. Danach wird die um eins erhöhte Sekundäradresse auf Band übertragen. Darauf folgend wird ab $C094 der Vorspann, bestehend aus Start- und Endadresse, File-Name und Füll-Bytes auf Band geschrieben. Zum Schluß folgt die Synchronisation 2, die zur Kennzeichnung mit einem Null-Byte abgeschlossen ist.

Ab $C0CB (PRGLOOP) folgt die eigentliche Programmspeicherroutine. Das Programm wird über den Programmstartvektor $AC/$AD indirekt - indiziert aus dem Speicher gelesen und an die Datasette übertragen. Danach wird der Vektor $AC/AD inkrementiert und mit dem Programmendevektor verglichen. Diese Schleife wird so lange durchlaufen, bis der Startvektor gleich dem Endvektor ist. Als Abschluß wird dann 255 mal die EXOR-Prüfsumme auf Band geschrieben. Nun wird 1 in das Motor-Flag geschrieben, um das Abschalten des Motors vorzubereiten, und das Speichern über die Betriebssystem-Routine in $FC93 beendet.

LADEROUTINE $C18E

Als erstes wird eine Synchronisation durch das Unterprogramm SSYNCH gesucht. Ist das letzte Byte der gefundenen Synchronisation, das in $AB geschrieben wurde, null, handelt es sich um eine Synchronisation 2, und es wird wieder zum Start dieser Routine gesprungen. Ist $AB ungleich null, wurde eine Synchronisation 1 gefunden, und der Wert in $AB ist die Sekundäradresse.

Ist das File mit der Sekundäradresse 1 abgespeichert oder ist die beim Ladebefehl übermittelte Sekundäradresse gleich eins, wird in ABSFLG Bit 1 gesetzt, und die Startadresse aus dem Kassettenpuffer wird in den Ladestartvektor geschrieben. Danach wird ab RELLOD FOUND NAME ausgegeben.

Das Unterprogramm ADTEST testet nun anhand von Bit 0 von ABSFLG, ob anstatt der im Kassettenpuffer stehenden Start-

adresse eine andere durch den L-Befehl übermittelte Adresse als Ladestartvektor benutzt werden soll. Ist Bit null von ABSFLG gesetzt, wird die mit L übermittelte Adresse, die in $AC/$AD steht, in den Startvektor geschrieben.

Entsprechend Bit zwei von ABSFLG wartet der Computer nun auf einen Tastendruck von Ihnen oder nicht. Nach dem Test, ob die <Stop>-Taste gedrückt wurde, wird der File-Name des gefundenen Files mit dem des gewünschten verglichen, falls ein Name übergeben wurde. Stimmen die Namen überein, kann der Ladevorgang beginnen, und das Status-Byte wird gelöscht. Durch die Addition der Programmlänge, gebildet aus der Differenz der Start- und Endadresse aus dem Vorspann, und dem Startvektor wird die Endadresse berechnet und in den Endvektor geschrieben. Das Unterprogramm PLOAD lädt dann das Programm.

Danach werden die Prüfsummen verglichen und ggf. der Status gesetzt. Wenn in ABSFLG kein Bit gesetzt ist, werden über die Betriebssytem-Routine STOEND die BASIC-Vektoren auf das Programmende gesetzt. Die Routine MERGE, LOAD und VERIFY laufen alle über die LOAD-Routine.

Bei MERGE wird der Startvektor entsprechend dem BASIC-Programmendevektor gesetzt. Danach wird relativ geladen. Bei VERIFY wird das VERIFY-Flag über das X-Register auf eins, bei LOAD auf null gesetzt. Vor der LOAD-Routine werden über GETPARA noch die weiteren Parameter eingelesen und in ABSFLG der entsprechende Wert gesetzt.

Datenspeicherung und Daten lesen

Die Daten werden als 3 Blöcke abgespeichert:

1. Block: Variablen

Bereich zwischen den Zeigern 45, 46/47, 48.

2. Block: Felder

Bereich zwischen den Zeigern 47, 48/49, 50.

3. Block: Strings

Bereich zwischen den Zeigern 51, 52/55, 56.

Die Länge der einzelnen Blöcke wird im Vorspann durch die Start- und Endadresse übergeben.

DATASAVE $C311

Ehe die Daten abgespeichert werden können, müssen sie auf minimalem Platz komprimiert werden. Deshalb wird zu Beginn die Garbage-Collection aufgerufen. Mit Hilfe von GETPARA wird nun der File-Name gesetzt. Danach wird als Sekundäradresse 1 gegeben, da ein bestimmter Speicherbereich abgespeichert werden soll. Durch ULOOP werden die Zeiger auf Anfangs- bzw. Endadressse des Variablenbereichs in die Start- und Endevektoren übertragen.

Zuletzt wird dieser Bereich durch das Unterprogramm ABSOLUT absolut abgespeichert. Die Felder und Strings werden danach analog zu den Variablen ohne File-Namen über ABSOLUT abgespeichert.

DATALOAD $C361

Zuerst wird über GETPARA der File-Name gesetzt. Durch Setzen des Bit 0 in ABSFLG wird erreicht, daß der Bereich an eine bestimmte Adresse geladen wird und die Adressen im Vorspann ignoriert werden. Als Ladeadresse wird die Variablen-Startadresse in den Startvektor übertragen. Über LOADR wird nun der Variablenbereich eingelesen. Danach wird zusätzlich Bit 2 in ABSFLG gesetzt, damit der Rechner nach dem Finden der nächsten Synchronisation 1 nicht auf einen Tastendruck des Bedieners wartet. Weiterhin wird kein File-Name gegeben. Der Inhalt von $AE, $AF setzt den Zeiger auf das Ende des beschriebenen Bereichs. Dieser Zeiger wird zugleich als Startvektor für die Felder in den Ladestartvektor übernommen, damit bei er-

neutem Aufruf von LOADR die Felder eingelesen werden. Danach wird $AE, $AF in den Zeiger auf das Ende der Felder geschrieben.

Da die String-Tabelle vom BASIC-RAM-Ende nach unten wächst, wird deren Ende nicht durch die BASIC-Programmlänge beeinflußt. Aus diesem Grunde wird dieser Bereich immer wieder dahin geladen, von wo er abgespeichert wurde. Darum wird Bit 0 in ABSFLG gelöscht. Danach wird die Tabelle durch die Routine LOADR geladen. Der Zeiger auf das untere Ende der String-Tabelle wird dann aus dem Kassettenpuffer gelesen und in 51, 52 gespeichert.

Zu guter Letzt wird der Status getestet und ggf. eine Fehlermeldung ausgegeben. Ist kein Fehler aufgetreten, wird der String-Descriptor zurückgesetzt und zum Hauptprogramm zurückgesprungen

10.11 Datenverarbeitung mit FastTape

In diesem Unterkapitel möchte ich Ihnen zeigen, daß auch mit einem Kassettenrekorder mit der geeigneten Software eine effiziente Datenverarbeitung möglich ist.

Das in diesem Unterkapitel abgedruckte Datenverarbeitungsprogramm hat folgende Spezifikationen:

- Daten speichern und laden mit FastTape. Je nach Größe der Datei dauert der Speicher- und Ladevorgang zwischen 20 Sekunden und 1,5 Minuten.
- Die Wahl der Datensatzanzahl ist Ihnen überlassen. Das Maximum ist von der freien Speicherkapazität des Rechners abhängig.
- Ein Datensatz kann in beliebig viele Felder unterteilt werden.
- Ein Feld darf maximal 80 Zeichen haben.
- Die Bezeichnung der einzelnen Felder ist frei wählbar und kann auch nach der ersten Festlegung geändert werden.

- Gesucht werden kann in einzelnen oder mehreren Feldern. Dabei können Sie wählen, ob ein oder alle Suchkriterien erfüllt werden sollen.
- Die Suchgeschwindigkeit ist in allen Feldern gleich.
- Es kann nach Übereinstimmung, Ungleichheit, größer und kleiner gesucht werden.

Falls Sie eine Befehlserweiterung mit einem INSTRING-Befehl besitzen, kann auch nach Teilbereichen innerhalb von Feldern gesucht werden.

- Wenn in mehreren Feldern gesucht wird, kann für jedes Feld eine andere Suchfunktion bestimmt werden.
- Es kann jederzeit eine HELP-Seite angezeigt werden, die die einzelnen Befehle erklärt.
- Sie können mit gefundenen Daten Listen erstellen, die Sie formatiert ausgeben können.
- Sie können frei bestimmen, welche Felder ausgegeben werden sollen.
- Die einzelnen Felder können mit oder ohne Feldbezeichnung ausgegeben werden.
- Listen können nach einem beliebigen Feld sortiert werden.

Damit Sie bei Bedarf das Programm an Ihre eigenen Bedürfnisse anpassen können, finden Sie am Ende dieses Unterkapitels eine detaillierte Programmbeschreibung.

Doch kommen wir nun zur Bedienung des Programms. Es ist möglich, mit diesem Programm eine Datei beliebiger Art einzurichten und zu bearbeiten. Sie können ein Literatur- oder Rezeptverzeichnis anlegen, Ihre Plattensammlung erfassen oder die Adressen Ihrer Freunde verwalten.

Um Ihnen die Funktion und Bedienung dieses Programms darzulegen, ist es am besten, wenn Sie, geleitet durch diese Bedienungsanleitung, eine Übungsdatei anlegen. Da ein Rezeptverzeichnis oder ähnliches zu diesem Zwecke zu umfangreich ist, legen Sie am besten eine Adreßdatei an. Ich gebe zu, daß dies schon eine äußerst abgedroschene Dateiart ist, doch ist es mit einer Adreßdatei möglich, mit wenig Schreibarbeit eine Datei

mit mehreren Datensätzen zu schreiben. Nach dem Starten mit RUN erscheint auf Ihrem Bildschirm das Menü.

```
***MENUE***
 1 Datei pflegen
 2 Blaettern
 3 Liste ausgeben
 4 Liste sortieren
 5 Datei laden
 6 Daten eingeben
 7 Daten speichern
 8 Datei einrichten
 9 Feldbez. aendern
 E   ENDE
```

Da Sie noch keine Datei angelegt haben, wählen Sie als erstes Punkt 9 des Menüs.

Datei einrichten

In diesem Menüpunkt müssen Sie die Grundcharakteristika Ihrer Datei eingeben. Sie werden gebeten, die Feldanzahl und die maximale Datensatzanzahl einzugeben. Diese Werte können nicht mehr geändert werden. Danach werden Sie aufgefordert, Feldbezeichnungen für die einzelnen Felder einzugeben. Für unsere Übungsadreßdatei geben Sie bitte als maximale Datensatzanzahl 10 oder mehr ein und als Feldanzahl 7. Die einzelnen Felder bezeichnen Sie am besten folgendermaßen:

1. Vorname
2. Name
3. Straße und Hausnummer
4. PLZ
5. Wohnort
6. Telefon
7. Hobby

Wenn Sie diese Eingaben getätigt haben, kehren Sie wieder zum Hauptmenü zurück. Ein nochmaliges Anwählen dieses Menüpunktes ist nun nicht mehr möglich.

Da Sie das Programm ja noch nicht richtig ausgetestet haben, es können sich immer einmal Fehler beim Abtippen einschleichen, speichern Sie diese Datei vorsichtshalber schon mal ohne Daten ab. Wählen Sie dafür den Punkt 7 "Datei speichern" an. Als erstes werden Sie nach dem File-Namen gefragt. Geben Sie "Adressen" ein, und drücken Sie <Return>.

Nun werden Sie gebeten, eine Datenkassette einzulegen. Legen Sie also eine Kassette als Datenkassette in Ihren Rekorder. Achten Sie dabei darauf, daß Sie die Kassette nicht ganz am Anfang stehen haben, damit die Datei auch sicher gespeichert wird. Drücken Sie <Return> und folgen der Aufforderung PRESS RECORD & PLAY ON TAPE.

Nachdem alle Variablen gespeichert wurden, kehrt der Rechner zum Menü zurück. Nun wollen wir einige Datensätze eingeben.

Daten eingeben

Wenn Sie diesen Menüpunkt angewählt haben, erscheint folgendes Bild:

```
        * DATEN EINGEBEN *

es existieren 0 Datensaetze

Vorname:
Name :
Strasse u. Hausnummer :
PLZ :
Wohnort :
Telefon :
Hobby :
```

Der Cursor steht unter der Feldbezeichnung "Vorname". Sie können nun in jedes Feld bis zu 80 Zeichen schreiben. So lange Sie innerhalb eines Feldes bleiben, steht Ihnen der volle Commodore-Bildschirmeditor zur Verfügung. Füllen Sie die Felder einmal folgendermaßen aus:

```
        * DATEN EINGEBEN *

es existieren 0 Datensaetze

Vorname:
Max
Name :
Meier
Strasse u. Hausnummer :
Oberstr. 3
PLZ :
4000
Wohnort :
Duesseldorf
Telefon :

Hobby :
Computer
```

Nachdem Sie die Eingabe im letzten Feld mit <Return> abgeschlossen haben, erscheint "RETURN" in der letzten Bildschirmzeile. Sie haben nun folgende Eingabemöglichkeiten:

<Return> Der Datensatz wird in die Datei übernommen, und Sie können den nächsten Datensatz eingeben.

<Q> Der Datensatz wird nicht übernommen, und das Programm kehrt zum Menü zurück.

<K> Der Cursor erscheint wieder im ersten Feld, und Sie können die Eingaben korrigieren.

Andere Taste
Der Datensatz wird übernommen, und das Programm kehrt zum Menü zurück.

Geben Sie also bei fehlerfreier Eingabe <Return>. Im Kopf wird nun angezeigt, daß ein Datensatz existiert, und der Cursor erscheint wieder im ersten Eingabefeld. In den Feldern steht nun noch die letzte Eingabe, die Sie jetzt ggf. überschreiben können. Falls sich einige Felder nicht ändern, können Sie durch Drücken von <Return> diesen Datensatz übernehmen. Geben Sie auf die oben beschriebene Weise einmal vier Adressen ein.

Falls eine Erweiterung mit INSTRING-Befehl zur Verfügung steht, geben Sie auch einmal den nächsten Datensatz ein:

```
         * DATEN EINGEBEN *

es existieren 4 Datensaetze

Vorname:
Marlies
Name :
Bange
Strasse u. Hausnummer :
Kaiserstr. 44
PLZ :
4150
Wohnort :
Krefeld
Telefon :
344322
Hobby :
Kochen/Computer
```

Kehren Sie nun durch Drücken einer beliebigen Taste zum Menü zurück.

Blättern

Wählen Sie nun Menüpunkt 2. In diesem Modus können Sie sich die Daten in der Reihenfolge, in der Sie sie eingegeben haben, ansehen und ggf. ändern. Durch Betätigen der <Return>-Taste können Sie jeweils um einen Datensatz weiterblättern. Mit der <->-Taste können Sie zurückblättern. Wenn Sie die <K>-Taste drücken, erscheint der Cursor im ersten Feld, und Sie können den angezeigten Datensatz ändern. Mit der <Q>-Taste kommen Sie wieder in das Menü.

```
         * * BLAETTERN * *

Datensatz Nr. 2

Vorname:
Peter
Name :
Namenlos
Strasse u. Hausnummer :
Dorfstr. 123
PLZ :
1234
```

```
Wohnort :
Nirgendwo
Telefon :
123456
Hobby :
Computer
```

Daten pflegen

Unter Menüpunkt 1 gelangen Sie in diesen Programmteil. Sie können hier die Daten selektieren, löschen, ändern und Listen erstellen.

Als erstes suchen wir im Feld "Name" alle Datensätze, in denen der Name mit "M" beginnt. Geben Sie also auf die Frage, nach welchen Feldern gesucht werden soll, eine 2 ein. Daraufhin erscheint die entsprechende Feldbezeichnung "invers" auf dem Bildschirm. In der vorletzten Bildschirmzeile erscheint:

```
1:T/2:=/3:<>/4:>/5:<
```

Das sind die Auswahlmöglichkeiten, die Sie haben:

1. Teil-String (nur mit INSTRING-Befehl)

Sie suchen nach einer Zeichenfolge, die an beliebiger Stelle innerhalb des Feldes stehen kann.

2. Gleich

Sie suchen nach einer signifikanten Zeichenfolge, die genau gleich Ihrem Such-String ist. Es werden dabei immer nur so viele Zeichen verglichen, wie Sie als Such-String eingeben. Geben Sie z.B. "Ma" ein, findet das Programm Mann, Maus, Mama etc. Dies gilt analog auch bei den folgenden Möglichkeiten:

3. Ungleich

Hier werden die Strings gefunden, die nicht mit dem Such-String übereinstimmen.

4. Größer oder gleich

Hier findet das Programm alle Strings, die größer oder gleich sind. Geben Sie z.B. "M" bei dem Namen ein, werden alle Namen gefunden, die mit M oder einem folgenden Buchstaben (N, O, P, Q...) beginnen.

5. Kleiner oder gleich

Geben Sie als Such-String "M" ein, werden alle Namen, die mit A - M beginnen, gefunden.

Drücken Sie "2", und Sie erhalten folgendes Bild:

```
              SUCHEN / AENDERN

 1. Vorname
 2. Name
 3. Strasse u. Hausnummer
 4. PLZ
 5. Wohnort
 6. Telefon
 7. Hobby

Nach welchen Feldern soll gesucht werden
 = Name
```

Wenn Sie innerhalb weiterer Felder suchen wollen, müssen Sie nur die entsprechende Zahl eingeben. Wir wollen aber jetzt erst einmal nur nach dem Namen suchen. Durch Drücken der <Return>-Taste schließen Sie die Eingabe ab und werden nun gefragt, ob ein Datensatz bei einer oder allen Übereinstimmungen gefunden werden soll. Da Sie ja sowieso nur ein Auswahlkriterium eingegeben haben, geben Sie <Return>.

Es erscheinen nun die bestimmte Feldbezeichnung und der Cursor. Da wir alle Namen suchen wollen, die mit "M" beginnen, geben Sie "M" ein und <Return>. Daraufhin erscheint in der letzten Bildschirmzeile wieder RETURN. Wenn Sie nun "K" drücken, können Sie die Eingabe korrigieren. Drücken Sie <Return>, beginnt der Rechner mit der Suche und zeigt dann

die Trefferzahl, die Datensatznummer des angezeigten Datensatzes und den Datensatz selbst an.

```
          SUCHEN /AENDERN

 3 Treffer
Datensatz Nr. 0

 Vorname:
 Max
 Name :
 Meier
 Strasse u. Hausnummer :
 Oberstr. 3
 PLZ :
 4000
 Wohnort :
 Duesseldorf
 Telefon :

 Hobby :
 Computer
          << RETURN >>
```

Sie können nun wie unter "Blättern" in den gefundenen Datensätzen blättern. Weiterhin können Sie sie durch <K> korrigieren und durch <L> löschen. Mit <Q> gelangen Sie wieder in das Menü.

Machen Sie sich jetzt am besten etwas mit den Menüpunkten 1 und 2 vertraut, damit Sie sehen, wie und wie schnell die gewünschten Datensätze gesucht und korrigiert werden können. Suchen Sie dann einmal nach einem Kriterium, das auf mehrere Datensätze zutrifft.

Wenn Sie so eine Liste aus mehreren Datensätzen erstellt haben, können Sie sie über Menüpunkt 4 nach einem beliebigen Feld, z.B. "Namen", sortieren. Über den Menüpunkt 3 können Sie die Liste auf dem Bildschirm oder auf einem Drucker ausgeben.

Liste ausgeben

In diesem Menüpunkt werden Sie gefragt, welche Felder ausgedruckt werden sollen. Durch die Eingabe der Feldindizes können Sie die gewünschten Felder wählen und mit <Return> die Eingabe beenden.

Wenn Sie statt dessen "K" eingeben, werden die angezeigten Felder gelöscht, und Sie können die Eingabe wiederholen. Nachdem Sie die gewünschten Felder gewählt haben, drücken Sie <Return>.

Als nächstes werden Sie nach einer Überschrift gefragt. Diese wird dann oberhalb Ihrer Liste ausgegeben. Bei der Druckerausgabe wird die Überschrift in Breitschrift ausgedruckt. Danach können Sie wählen, ob die Feldbezeichnungen mit ausgedruckt werden sollen. Als letztes wird danach gefragt, ob die Ausgabe auf den Bildschirm oder über einen Drucker erfolgen soll.

Das Programm ist so konzipiert, daß jedes Feld in eine eigene Zeile geschrieben wird. Durch Programmänderungen in den Zeilen 2.870-2.930 können Sie das Ausdruckformat entsprechend Ihren Wünschen ändern. Wenn Sie die Liste auf den Bildschirm ausgeben, können Sie die Ausgabe durch Drücken der <Space>-Taste anhalten und wieder starten. Ich hoffe, daß Ihnen die Bedienung dieses Programms nach einigem Üben keine Schwierigkeiten mehr macht.

```
PROGRAMM:DATEIVERW

1   GOTO 1060
90  REM "< 4 SPACE>SUCH-ROUTINE"
100  IF (F > FE) OR (N >  = AS) THEN  RETURN
110 A = SF(F)
20  ON A GOSUB 150,190,250,280,220
130 F = F + 1:  GOTO 100
140 :  REM "< 2 SPACE>TEILBEREICH
150 T =  RENAME (U$(F),S$(F))
160  IF T <  > 0 THEN N = N + 1
170  RETURN
180 :  REM "< 2 SPACE>GANZ '='
190  IF S$(F) =  LEFT$ (U$(F), LEN (S$(F))) THEN N = N + 1
200  RETURN
210 :  REM "< 2 SPACE>GANZ '<='
220  IF S$(F) >  =  LEFT$ (U$(F), LEN (S$(F))) THEN N = N + 1
```

```
230   RETURN
240  :   REM  "< 2 SPACE>GANZ '<>'
250   IF S$(F) <  >  LEFT$ (U$(F), LEN (S$(F))) THEN N = N + 1
260   RETURN
270  :   REM  "< 2 SPACE>GANZ '>='
280   IF S$(F) <  =  LEFT$ (U$(F), LEN (S$(F))) THEN N = N + 1
290   RETURN
1000   REM  "WARTESCHLEIFE
1010   POKE 198,0:  WAIT 198,1:  GET A$: A =  ASC (A$):  REM
"WARTESCHLEIFE
1020   IF A = 133 THEN  GOSUB 3750
1030   RETURN
1050   REM "< 2 SPACE>INITIALISIERUNG
1060  DW$ = "<HOME>< 25 CRSR DOWN>"
1070   SYS 65517: SP =  PEEK (781): ZE =  PEEK (782):  REM  "
BILDSCHIRMDATEN HOLEN
1080   C1 = 53281: C2 = 1:  IF SP = 22 THEN C1 = 36879: C2 = 8
1090   DIM Z$(4): Z$(0) = " T": Z$(1) = " =": Z$(2) = "<>": Z$(3) =
">=": Z$(4) = "<="
1100   REM  "< 6 SPACE>HAUPTPROGRAMM
1110   PRINT  CHR$ (14):  POKE C1,6 * C2
1120   GOSUB 3560:  PRINT :  PRINT : TA =  INT (SP / 5) - 2
1130   PRINT  TAB( TA)"<WHT><RVS ON>1<RVS OFF> DATEI PFLEGEN"
1140   PRINT  TAB( TA)"<RVS ON>2<RVS OFF> BLAETTERN "
1150   PRINT  TAB( TA)"<RVS ON>3<RVS OFF> LISTE AUSGEBEN"
1160   PRINT  TAB( TA)"<RVS ON>4<RVS OFF> LISTE SORTIEREN"
1170   PRINT  TAB( TA)"<RVS ON>5<RVS OFF> DATEN EINGEBEN "
1180   PRINT  TAB( TA)"<RVS ON>6<RVS OFF> DATEI LADEN< 2 SPACE>"
1190   PRINT  TAB( TA)"<RVS ON>7<RVS OFF> DATEI SPEICHERN"
1200   PRINT  TAB( TA)"<RVS ON>8<RVS OFF> DATEI EINRICHTEN"
1210   PRINT  TAB( TA)"<RVS ON>9<RVS OFF> FELDBEZ. AENDERN"
1220   PRINT  TAB( TA)"<RVS ON>E<RVS OFF>< 4 SPACE><RVS ON>ENDE<RVS
OFF>"
1230   GOSUB 1010: A =  VAL (A$):  IF A$ = "E" THEN 1270
1240   IF (A < S) AND (FL AND 4) = 0 THEN 1110
1250   ON A GOSUB 2130,2000,2700,2990,1860,1570,1460,1700,1780
1260   GOTO 1110
1265   REM  "PROGRAMM BEENDEN
1270   IF(FL AND 2)=2 OR (FL AND 4)=0 THEN END
1280   PRINT "<CLR HOME>< 20 CRSR DOWN>":  PRINT  TAB( SP / 2 -
8)"<WHT>TROTZDEM ENDEN":  GOSUB 1310
1290   IF FL <  > 255 THEN 1110
1300   END
1305   REM  "< 9 SPACE>Fehler
1310   POKE C1,5 * C2
1320   PRINT "<HOME>":  PRINT "< 7 CRSR DOWN>"
1330   PRINT :  PRINT  TAB( SP / 2 - 3)"<RED><RVS ON>Datei"
1340   PRINT :  PRINT  TAB( SP / 2 - 6)"<RVS ON>wurde nicht"
1350   PRINT :  PRINT  TAB( SP / 2 - 7)"<RVS ON>abgespeichert"
1360   FOR I = 0 TO 15:  GET A$:  IF A$ <  > "" THEN I = 15:  NEXT :
GOTO 1440
1370   NEXT
1380   PRINT "<HOME>":  PRINT "< 7 CRSR DOWN>"
```

```
1390   PRINT :  PRINT  TAB( SP / 2 - 3)"<RED>Datei"
1400   PRINT :  PRINT  TAB( SP / 2 - 6)"wurde nicht"
1410   PRINT :  PRINT  TAB( SP / 2 - 7)"abgespeichert"
1420   FOR I = 0 TO 15:  GET A$:  IF A$ = "" THEN I = 15:  NEXT :  GOTO
1310
1430   NEXT
1440   IF A$ = "J" THEN FL = 255
1450   RETURN
1460   REM  "< 8 SPACE>SPEICHERN
1470   IF (FL AND 1) = 0 THEN  RETURN
1480   POKE C1,7 * C2:  PRINT "<BLU>":  GOSUB 3590
1490   PRINT :  PRINT :  PRINT "< 2 SPACE>"F$"<CRSR UP>":  INPUT "FILE-
NAME";F$
1500   PRINT :  PRINT "DATENCASSETTE EINLEGEN"
1510   PRINT :  PRINT :  PRINT "< 3 SPACE><< Return<SHIFT SPACE>>>"
1520   GOSUB 1010
1530   IF A$ = "K" THEN  PRINT "<HOME>":  GOTO 1490
1540   IF A$ = "Q" THEN  RETURN
1550   IF A <  > 13 THEN 1520
1560  ←DSF$: FL = FL OR 2:  RETURN
1570   REM  "< 9 SPACE>LADEN
1580   IF (FL AND 6) = 4 THEN  GOSUB 1680:  IF FL <  > 255 THEN  RETURN
1590   POKE C1,7 * C2:  PRINT "<BLU>":  GOSUB 3600
1600   PRINT :  PRINT :  PRINT "< 2 SPACE>"F$"<CRSR UP>":  INPUT "FILE-
NAME";F$
1610   PRINT :  PRINT "DATENCASSETTE EINLEGEN"
1620   PRINT :  PRINT :  PRINT "< 3 SPACE><< Return<SHIFT SPACE>>>"
1630   GOSUB 1010
1640   IF A$ = "K" THEN  PRINT "<HOME>":  GOTO 1600
1650   IF A$ = "Q" THEN  RETURN
1660   IF A <  > 13 THEN 1630
1670  ←DLF$: FL = FL AND 251:  RETURN
1680   PRINT "<CLR HOME>< 20 CRSR DOWN>":  PRINT  TAB( SP / 2 -
8)"<WHT>TROTZDEM LADEN"
1690   GOSUB 1310:  RETURN
1700   REM  "< 5 SPACE>DATEI ERSTELLEN
1710   POKE C1,8 * C2:  IF FL THEN  RETURN
1720   GOSUB 3550:  REM  " TITEL SCHREIBEN
1730   PRINT :  PRINT :  INPUT "FELDANZAHL";FA
1740   PRINT :  INPUT "<CRSR DOWN>MAX DATENSAETZE";RA
1750   PRINT :  PRINT "<< Return >> ":  GOSUB 1010:  IF A <  > 13 THEN
PRINT "<HOME>":  GOTO 1730
1760   DIM M$(FA),E$(FA,RA),SF(FA),S$(FA),U$(FA),GF%(RA)
1770  FL = 1:  REM  "< 2 SPACE>FLAG SETZEN
1775   REM  "< 4 SPACE>FELDBEZEICHNUNGEN EINGEBEN
1780   IF (FL AND 1) = 0 THEN  RETURN
1790   GOSUB 3550:  REM  "UEBERSCHRIFT SCHREIBEN
1800   PRINT :  PRINT "GEBEN SIE DIE ";:  IF SP < 40 THEN  PRINT :
PRINT
1810   PRINT "FELDBEZEICHNUNGEN EIN"
1820   PRINT :  FOR I = 0 TO FA - 1:  PRINT :  PRINT "<RVS ON>FELD "I +
1"<RVS OFF>"
1830   PRINT "< 2 SPACE>"M$(I)"<CRSR UP>":  INPUT M$(I):  NEXT
```

```
1840  PRINT  LEFT$ (DW$,ZE)"<< Return >>";:  GOSUB 1010:  IF A$ = "K"
THEN  PRINT "<HOME>":  GOTO 1800
1850  FL = FL OR 4:  RETURN
1860  REM  "< 5 SPACE>DATEN EINGEBEN
1865  IF FL AND 1 = 0 THEN  RETURN
1870  POKE C1,3 * C2:  PRINT "<BLU>"
1880  CLOSE 1:  OPEN 1,0
1890  GOSUB 3580:  PRINT :  PRINT "ES EXISTIEREN"RT;:  IF SP < 40 THEN
PRINT
1900  PRINT "DATENSAETZE"
1910  R1 = RT: R2 = RT:  IF (RT > 0) AND (A$ <  > "K") THEN R1 = RT - 1
1920  GOSUB 3650:  GOSUB 3670:  REM  "< 2 SPACE>FELDER ANZEIGEN
1930  PRINT  LEFT$ (DW$,ZE)"<GRN>< 3 SPACE><< Return<SHIFT SPACE>>>
<WHT>";:  GOSUB 1010
1940  IF A$ = "Q" THEN  RETURN
1950  IF A$ = "K" THEN 1890
1960  IF A <  > 13 THEN FL = FL OR 12: RT = RT + 1:  RETURN
1970  RT = RT + 1:  IF RT < RA THEN 1890
1980  GOSUB 3580:  PRINT :  PRINT :  PRINT "MAXIMALE SATZZAHL ";:  IF
SP < 40 THEN  PRINT
1990  PRINT :  PRINT "ERREICHT":  GOSUB 1010:  RETURN
2000  REM  "< 7 SPACE>BLAETTERN
2005  IF (FL AND 8) = 0 THEN  RETURN
2010  POKE C1,6 * C2:  PRINT "<WHT>"
2020  R1 = 0
2025  IF R1 < 0 THEN R1 = R1 + RT
2030  GOSUB 3610:  PRINT :  PRINT "DATENSATZ NR."R1
2040  GOSUB 3650
2050  PRINT  LEFT$ (DW$,ZE)"<GRN>< 3 SPACE><< Return<SHIFT SPACE>>>
<WHT>";:  GOSUB 1010
2060  IF A$ = "-" THEN R1 = R1 - 1:  GOTO 2025
2070  IF A$ = "Q" THEN  RETURN
2080  IF A$ = "A" THEN  GOSUB 2560:  GOTO 2060
2090  IF A <  > 13 THEN 2050
2100  IF R1 < RT - 1 THEN R1 = R1 + 1:  GOTO 2030
2110  GOSUB 3610:  PRINT "< 8 CRSR DOWN><CRSR RIGHT><RVS ON> KEINE
WEITEREN DATEN "
2120  R1 = 0:  GOTO 2050
2130  REM  "< 5 SPACE>DATEI PFLEGEN
2135  IF (FL AND 8) = 0 THEN  RETURN
2140  POKE C1,4 * C2
2150  FOR I = 0 TO FA - 1: SF(I) = 0:  NEXT : AS = 0
2160  GOSUB 3620:  FOR I = 0 TO FA - 1:  PRINT I + 1"<CRSR LEFT>.
"M$(I):  NEXT
2170  PRINT :  PRINT "NACH WELCHEN FELDERN ";:  IF SP < 40 THEN  PRINT
:  PRINT
2180  PRINT "SOLL GESUCHT WERDEN":  PRINT :  PRINT
2190  GOSUB 1010:  IF A = 13 THEN 2262
2200  IF A$ = "Q" THEN  RETURN
2210  IF A$ = "K" THEN 2150
2220  I =  VAL (A$):  IF I < 1 OR I > FA THEN 2190
2230  IF SF(I - 1) THEN 2190
2240  D6 =  PEEK (214):  REM  " CURSOR-ZEILE RETTEN
```

```
2250  SF(I - 1) = 1:  PRINT  TAB( 3)"<RVS ON>"M$(I - 1):  GOSUB 3710
2260   PRINT  LEFT$ (DW$,D6 + 1)Z$(SF(I - 1) - 1): AS = AS + 1:  GOTO
2190
2262   GOSUB 3620:  PRINT :  PRINT "FINDEN BEI"
2264   PRINT :  PRINT  TAB( 4)"1 = EINER"
2265   PRINT :  PRINT  TAB( 4)"0 = ALLEN"
2266   PRINT :  INPUT "UEBEREINSTIMMUNGEN";UE
2267   IF UE THEN AS = UE
2270   GOSUB 3620:  CLOSE 1:  OPEN 1,0
2280   PRINT :  PRINT "GEBEN SIE SUCHSTRINGS ";:  IF SP < 40 THEN  PRINT
: PRINT
2290   PRINT "EIN":  PRINT :  PRINT
2300   FOR F = 0 TO FA - 1:  IF SF(F) = 0 THEN 2330
2310   PRINT "<RVS ON>"M$(F)" :  "
2320   PRINT S$(F)"<CRSR UP>":  INPUT# 1,S$(F):  PRINT
2330   NEXT
2340   PRINT  LEFT$ (DW$,ZE)"<< Return >>";:  GOSUB 1010:  IF A$ = "K"
THEN  PRINT "<HOME>":  GOTO 2280
2350   IF A$ = "Q" THEN  RETURN
2360   IF A <  > 13 GOTO 2340
2370   REM  "< 9 SPACE>SUCHEN
2372   FOR F = 0 TO FA - 1:  IF SF(F) THEN FB = F: F = FA
2374   NEXT :  REM  "NIEDRIGSTES UND HOECHSTES
2376   FOR F = FA - 1 TO 0 STEP  - 1:  IF SF(F) THEN FE = F: F = 0
2378   NEXT :  REM  "SUCHFELD FESTSTELLEN
2380  Z = 0:  FOR  R  = 0  TO  RT - 1
2390   FOR F = FB TO FE - 1:  IF SF(F) THEN U$(F) = E$(F,R)
2400   NEXT :  REM  "UNTERSUCHSTRINGS UMSPEICHERN
2410  N = 0: F = FB:  GOSUB 100:  IF N >  = AS THEN GF%(Z) = R: Z = Z +
1
2420   NEXT :  IF Z = 0 THEN 2130
2430   REM  "< 5 SPACE>DATENSAETZE ANZEIGEN
2440  I = 0
2450   GOSUB 3620:  PRINT :  PRINT Z" TREFFER":  PRINT "DATENSATZ
NR."GF%(I)
2460  R1 = GF%(I):  GOSUB 3650
2470   PRINT  LEFT$ (DW$,ZE)"<GRN>< 3 SPACE><< Return<SHIFT SPACE>>>
<WHT>";:  GOSUB 1010
2480   IF A$ = "-" AND I > 0 THEN I = I - 1:  GOTO 2450
2490   IF A$ = "Q" THEN  RETURN
2500   IF A$ = "A" THEN  GOSUB 2560:  GOTO 2480
2510   IF A$ = "L" THEN 2640
2520   IF A <  > 13 THEN 2470
2530   IF I < Z - 1 THEN I = I + 1:  GOTO 2450
2540   GOSUB 3620:  PRINT "< 8 CRSR DOWN>< 3 CRSR RIGHT><RVS ON> KEINE
WEITEREN DATEN "
2550   PRINT  LEFT$ (DW$,ZE)"<GRN>< 3 SPACE><< Return<SHIFT SPACE>>>
<WHT>";:  GOSUB 1010:  GOTO 2450
2560   REM  "< 9 SPACE>AENDERN
2570   CLOSE 1:  OPEN 1,0: R2 = R1:  GOSUB 3670
2580   PRINT  LEFT$ (DW$,ZE)"<GRN>< 3 SPACE><< Return<SHIFT SPACE>>>
<WHT>";:  GOSUB 1010
2600   IF A$ = "K" THEN 2570
```

```
2610  FL = FL OR 4: FL = FL AND 253:  RETURN
2620   RETURN
2630   REM  "< 8 SPACE>LOESCHEN
2640   FOR I1 = GF%(I) + 1 TO RT - 1:  REM  " DATEN
2650   FOR J = 0 TO FA - 1:  REM    "< 7 SPACE>UMSPEICHERN
2660  E$(J,I1 - 1) = E$(J,I1)
2670   NEXT J,I1
2680  Z = Z - 1: RT = RT - 1: FL = FL OR 4:  GOTO 2450
2690   REM  "< 5 SPACE>LISTE AUSGEBEN
2695   IF Z = 0 THEN  RETURN
2700   POKE C1,7 * C2:  PRINT "<BLU>"
2710   FOR I = 0 TO FA - 1: SF(I) = 0:  NEXT
2720   GOSUB 3630
2725   FOR I = 0 TO FA - 1:  PRINT I + 1"<CRSR LEFT>. "M$(I):  NEXT
2730   PRINT :  PRINT "WELCHE FELDER SOLLEN ";:  IF SP < 40 THEN  PRINT
:  PRINT
2740   PRINT "AUSGEDRUCKT WERDEN":  PRINT :  PRINT
2750   GOSUB 1010:  IF A = 13 THEN 2810
2760   IF A$ = "Q" THEN  RETURN
2770   IF A$ = "K" THEN 2710
2780  I =  VAL (A$):  IF I < 1 OR I > FA THEN 2750
2790   IF SF(I - 1) THEN 2750
2800  SF(I - 1) = 1:  PRINT "<RVS ON>"M$(I - 1):  PRINT :  GOTO 2750
2810   PRINT :  INPUT "UEBERSCHRIFT";UB$
2820   PRINT :  PRINT "FELDBEZEICHNUNGEN":  PRINT :  INPUT "AUSDRUCKEN
<COMM :>1/0<COMM ;>";FM$
2830  FM =  VAL (FM$)
2840   PRINT :  INPUT "DRUCKER=4/BILDS.=0";DV
2850   IF DV THEN  OPEN 4,DV,7:  CMD 4
2860   PRINT  CHR$ (14)UB$ CHR$ (15):  PRINT :  PRINT
2870   FOR R = 0 TO Z - 1
2880   FOR F = 0 TO FA - 1:  IF SF(F) = 0 THEN 2910
2890   IF FM THEN  PRINT :  PRINT "<RVS ON>"M$(F)" :  <RVS OFF>"
2900   PRINT E$(F,GF%(R))
2910   NEXT :  IF DV THEN 2930
2920   GET A$:  IF A$ <  > "" THEN  POKE 198,0:  WAIT 198,1:  GET A$
2930   PRINT :  PRINT :  NEXT
2940   IF DV THEN  PRINT# 4:  CLOSE 4:  RETURN
2950   PRINT :  PRINT "< 4 SPACE><< Return<SHIFT SPACE>>>"
2960   GOSUB 1010:  IF A <  > 13 THEN 2960
2970   RETURN
2980   REM  "< 8 SPACE>SORTIEREN
2990   IF Z = 0 THEN  RETURN
3000   GOSUB 3640
3010   FOR I = 0 TO FA - 1:  PRINT I + 1"<CRSR LEFT>. "M$(I):  NEXT
3020   PRINT "<CRSR DOWN>NACH WELCHEM FELD ";:  IF SP < 40 THEN  PRINT :
PRINT
3030   INPUT "SOLL SORTIERT WERDEN";SF: SF = SF - 1
3040  F1 = 0: FH = 0: FH =  PEEK (49) +  PEEK (50) * 256:  REM
"OBERGRENZE RETTEN
3050  F1 =  PEEK (47) +  PEEK (48) * 256:  REM  " UNTERGRENZE RETTEN
3060   DIM N$(Z),Z(Z + 1)
3070  Q = 1:  FOR I = 0 TO Z - 1:  REM  "< 4 SPACE>SORTIERFELD
```

```
3080 N$(Q) = E$(SF,GF%(I)):  REM  " UMSPEICHERN
3090 Z(Q) = Q: Q = Q + 1:  NEXT
3100  GOSUB 3240:  REM  "< 9 SPACE>SORTIERROUTINE
3110  FOR X = 0 TO FA - 1
3130  FOR Y = 0 TO Z - 1
3140 N$(Y + 1) = E$(X,GF%(Y))
3160  NEXT
3170  FOR Y = 0 TO Z - 1
3180 E$(X,GF%(Y)) = N$(Z(Y + 1))
3200  NEXT Y,X
3210  FOR I = 0 TO Z - 1: N$(I) = "":  NEXT :  REM  "STRINGS VON N$()
LOESCHEN
3220  GOSUB 3520:  REM  "WIEDERHERSTELLUNG DER ALTEN ZEIGER
3230  RETURN
3235  REM  "< 4 SPACE>SORTIERROUTINE
3240 TI$ = "000000":  PRINT "<CLR HOME>"
3250 L =  INT (Z / 2) + 1
3260 R = Z:  PRINT "<CLR HOME>"R"<CRSR LEFT>< 3 SPACE>"
3270  IF L > 1 THEN 3340
3280  IF R <  = 1 THEN 3330
3290 H2$ = N$(L): H1 = Z(L)
3300 N$(L) = N$(R): Z(L) = Z(R)
3310 N$(R) = H2$: Z(R) = H1
3320 R = R - 1
3330  GOTO 3350
3340 L = L - 1
3350 J = L
3360 I = 2 * J
3370 H2$ = N$(J): H1 = Z(J)
3380  IF I > R THEN 3480
3390  IF I >  = R THEN 3420
3400  IF N$(I) >  = N$(I + 1) THEN 3420
3410 I = I + 1:  PRINT "<CLR HOME>"I"<CRSR LEFT>< 3 SPACE>"
3420  IF I > R THEN 3480
3430  IF H2$ >  = N$(I) THEN 3480
3440 N$(J) = N$(I): Z(J) = Z(I)
3450 J = I
3460 I = 2 * J
3470  GOTO 3390
3480 N$(J) = H2$: Z(J) = H1
3490  IF R <  > 1 THEN 3270
3500  PRINT "<CLR HOME>< 2 CRSR DOWN>SORTIERZEIT";TI$
3510  RETURN
3515  REM  "POINTER WIEDERHERSTELLEN
3520 J =  PEEK (47) +  PEEK (48) * 256 + (FH - F1)
3530  POKE 49,J AND 255:  POKE 50, INT (J / 256)
3540  RETURN
3545  REM  "< 4 SPACE>UEBERSCHRIFTEN
3550  PRINT "<CLR HOME>";:  PRINT  TAB( SP / 2 - 10)"* Datei<SHIFT
SPACE>erstellen *":  RETURN
3560  PRINT "<CLR HOME>";:  PRINT  TAB( SP / 2 - 10)"<WHT>* * * M<SHIFT
SPACE>E<SHIFT SPACE>N<SHIFT SPACE>U<SHIFT SPACE>E<SHIFT SPACE>*<SHIFT
SPACE>* *":  RETURN
```

```
3570  PRINT "<CLR HOME>";:  PRINT  TAB( SP / 2 - 9)"* Datei<SHIFT
SPACE>pflegen *":  RETURN
3580  PRINT "<CLR HOME><BLU>";:  PRINT  TAB( SP / 2 - 10)"* Daten<SHIFT
SPACE>eingeben *":  RETURN
3590  PRINT "<CLR HOME>";:  PRINT  TAB( SP / 2 - 8)"* Daten<SHIFT
SPACE>speichern *":  RETURN
3600  PRINT "<CLR HOME>";:  PRINT  TAB( SP / 2 - 6)"* Daten<SHIFT
SPACE>laden *":  RETURN
3610  PRINT "<CLR HOME>";:  PRINT  TAB( SP / 2 - 8)"* * Blaettern<SHIFT
SPACE>* *":  RETURN
3620  PRINT "<CLR HOME>";:  PRINT  TAB( SP / 2 - 8)"Suchen / Aendern":
RETURN
3630  PRINT "<CLR HOME>";:  PRINT  TAB( SP / 2 - 7)"Liste<SHIFT
SPACE>ausgeben":  RETURN
3640  PRINT "<CLR HOME>";:  PRINT  TAB( SP / 2 - 7)"Liste<SHIFT
SPACE>sortieren":  RETURN
3645  REM  "< 4 SPACE>FELDER AUSGEBEN
3650  PRINT "<HOME>< 5 CRSR DOWN>":  FOR F = 0 TO FA - 1:  PRINT "<RVS
ON>"M$(F)" :  "
3660  PRINT E$(F,R1):  NEXT :  RETURN
3670  PRINT "<HOME>< 5 CRSR DOWN>":  FOR F = 0 TO FA - 1:  PRINT "<RVS
ON>"M$(F)" :  "
3680  PRINT E$(F,R1)"<CRSR UP>": A =  LEN (E$(F,R1))
3690  IF A > SP THEN  FOR X = 0 TO A / SP:  PRINT "<CRSR UP>";:  NEXT
3700  INPUT# 1,E$(F,R2):  PRINT :  NEXT :  RETURN
3705  REM  "< 3 SPACE>SUCHFORM WAEHLEN
3710  PRINT  LEFT$ (DW$,ZE - 1)"<RVS ON>1: ■/2: =/3: <>/4: >/5: <<RVS
OFF>";
3720  GOSUB 1010: J =  VAL (A$):  IF J < 1 OR J > 5 THEN 3720
3730 SF(I - 1) = J
3740  PRINT  LEFT$ (DW$,ZE - 1)"< 20 SPACE>";:  RETURN
3750  REM  "< 6 SPACE>■ELPSEITE
3760  PRINT "<CLR HOME>* * * * * * HELP-SEITE * * * * *"
3770  PRINT :  PRINT "WENN< 2 SPACE><< Return >> ERSCHEINT HABEN SIE"
3780  PRINT :  PRINT "OPTIONEN : ":  PRINT :  PRINT
3790  PRINT "Return< 2 SPACE>= WEITERMACHEN":  PRINT
3800  PRINT "K< 7 SPACE>= LETZTE EINGABE WIEDERHOLEN":  PRINT
3810  PRINT "Q< 7 SPACE>= PROGRAMMPUNKT ABBRECHEN":  PRINT
3820  PRINT "-< 7 SPACE>= EINEN SATZ ZURUECK":  PRINT
3830  PRINT "A< 7 SPACE>= EINEN SATZ AENDERN":  PRINT
3840  PRINT "L< 7 SPACE>= EINEN SATZ LOESCHEN":  PRINT
3850  GET A$:  IF A$ = "" THEN 3850
3860  RETURN
5000  "************************************
5010  "*< 2 SPACE>DATEIVERWALTUNG MIT FASTTAPE< 2 SPACE>*
5020  "*< 6 SPACE>(C) DIRK PAULISSEN< 8 SPACE>*
5030  "*< 32 SPACE>*
5040  "* VERWENDETE VARIABLEN : < 9 SPACE>*
5050  "* FA< 4 SPACE>= FELDANZAHL< 13 SPACE>*
5060  "* RA< 4 SPACE>= RECORDANZAHL(MAX)< 6 SPACE>*
5070  "* RT< 4 SPACE>=< 3 SPACE>- '' - (AKTUELL)< 5 SPACE>*
5080  "* F< 5 SPACE>= AKTUELLES FELD< 9 SPACE>*
5090  "* R< 5 SPACE>= AKTUELLER RECORD< 7 SPACE>*
```

```
5100  "* SP< 4 SPACE>= SPALTENANZAHL< 10 SPACE>*
5110  "* ZE< 4 SPACE>= ZEILENZAHL< 13 SPACE>*
5120  "* A$< 4 SPACE>= FRAGESTRING< 12 SPACE>*
5130  "* FL< 4 SPACE>= FLAG< 19 SPACE>*
5140  "* E$(F,R) = DATENSTRING< 10 SPACE>*
5150  "* SF(F)< 3 SPACE>= SUCHFELD-FLAG< 9 SPACE>*
5160  "*< 9 SPACE>1 :  TEILSTRING SUCHEN< 2 SPACE>*
5170  "*< 9 SPACE>2 :  GANZ '='< 11 SPACE>*
5180  "*< 9 SPACE>3 :  GANZ '<>'< 10 SPACE>*
5190  "*< 9 SPACE>4 :  GANZ '>='< 10 SPACE>*
5200  "*< 9 SPACE>5 :  GANZ '<='< 10 SPACE>*
5220  "* S$(F)< 3 SPACE>= SUCHSTRING< 11 SPACE>*
5230  "* U$(F)< 3 SPACE>= UNTERSUCHSTRING< 6 SPACE>*
5240  "* GF%()< 3 SPACE>= NUMMER DER GEFUNDE-< 2 SPACE>*
5250  "*< 11 SPACE>NEN DATENSAETZE< 6 SPACE>*
5300  "************************************
```

10.11.1 Programmbeschreibung

Der Programmaufbau ist folgender:

```
  90 - 1030   Schnelle Unterprogramme
1050 - 1090   Initialisierung
1100 - 1260   Hauptprogramm (Menü)
1270 - 3540   Unterprogramme, die vom Menü aufgerufen werden
3550 - 3640   Überschriften
3650 - 3850   Unterprogramme, die von anderen Unterprogrammen
              aufgerufen werden
5000 - Ende   Copyright-Vermerk und Aufzählung der benutzten Variablen
```

Arbeitsweise

Das Hauptprogramm besteht nur aus dem Menü und der Verzweigung in das gewählte Unterprogramm. Zur Datensicherheit wurde ein Flag "fl" eingeführt, welches immer am Anfang eines Unterprogramms getestet und innerhalb der Unterprogramme gesetzt wird. Die einzelnen Bits von fl haben folgende Bedeutung, wenn sie gesetzt sind:

Bit	Wert	Bedeutung
0	1	Datei eingerichtet
1	2	Datei gespeichert
2	4	Datei verändert
3	8	Datei enthält Datensätze

Wird versucht, eine neue Datei zu laden oder das Programm zu beenden, ehe eine geänderte Datei abgespeichert wurde, wird in die Fehlerroutine 1305 bis 1450 verzweigt. Die Programmteile "Laden" und "Speichern" verstehen sich von selbst.

Datei erstellen (1700 - 1850)

Nachdem der Titel durch Zeile 1720 geschrieben wurde, wird per INPUT die Feldanzahl (fa) und Datensatzanzahl (ra) eingelesen. Durch die Abfrage in 1750 wird es ermöglicht, eine fehlerhafte Eingabe zu korrigieren. Entsprechend "fa" und "ra" werden in 1760 die Felder dimensioniert und in 1770 Bit 0 in "fl" gesetzt.

In Zeile 1780 wird Bit 0 in "fl" getestet, da die Routine zur Eingabe und Änderung der Feldbezeichnungen auch vom Menü angesprungen werden kann. In Zeile 1800 und 1810 wird der auszugebene String entsprechend der Spaltenzahl des Computers in einer oder zwei Zeilen ausgedruckt.

Innerhalb einer Schleife (Zeile 1820 - 1830) werden die einzelnen Feldbezeichnungen in m$(fa) eingelesen. Eventuell vorhandene werden auf dem Bildschirm ausgegeben und können durch <Return> übernommen werden. Durch die Abfrage in Zeile 1840 ist wieder eine Korrektur möglich.

Daten eingeben (1860 - 1990)

In Zeile 1865 wird "fl" daraufhin getestet, ob schon eine Datei erstellt wurde. In der Zeile 1880 wird durch OPEN 1,0 die Tastatur eröffnet. Das hat zur Folge, daß bei einem folgenden INPUT#1 kein Fragezeichen ausgegeben wird. Aus Sicherheitsgründen wird vor jedem OPEN-Befehl die entsprechende Datei geschlossen.

Über die Zeilen 1890-1900 wird die Überschrift und, entsprechend der Spaltenzahl, ein String ausgegeben. In den Unterprogrammen, welche die einzelnen Felder eines Satzes anzeigen (3650-3660) bzw. neue Felder einlesen (3670-3700), wird "r1" bzw. "r2" als Satzzeiger benutzt. Aus diesem Grunde wird in

Zeile 1910 der Zeiger auf den einzugebenden Satz "rt" in "r1" und "r2" übertragen. Wurde schon ein Satz eingegeben (rt größer null), wird "r1" dekrementiert, so daß die Felder des zuletzt eingegebenen Satzes angezeigt werden. Ab Zeile 1930 wird abgefragt, ob abgebrochen (a$ = "q"), korrigiert (a$ = "k"), beendet (a ungleich 13) oder weitergemacht werden soll. Ist a = 13, wird in Zeile 1970 getestet, ob die maximale Satzzahl erreicht wurde.

Blättern (2000)

Mit Hilfe der Laufvariablen "r1" werden über das Unterprogramm "Felder ausgeben" die einzelnen Datensätze angezeigt und dann die Tastatur über die "Warteschleife" (1010) abgefragt. Entsprechend der Eingabe wird weitergeblättert (2100), zum Menü zurückgekehrt (2070) oder in das Unterprogramm "Ändern" (2080) verzweigt.

Datei pflegen (2130)

Nach Abfrage von "fl" und Bildschirm-POKE werden die Such-Flags (sf(i)) auf null gesetzt (2150). Über die "Warteschleife" wird ab Zeile 2190 abgefragt, ab welchem Feld gesucht werden soll. Wurde eine gültige, noch nicht gewählte Feldnummer eingegeben (der Test dafür ist in Zeile 2220-2230), wird in Zeile 2240 die aktuelle Cursor-Zeile in "d6" gerettet und in Zeile 2250 die Feldbezeichnung des gewünschten Feldes ausgedruckt.

Über das Unterprogramm "Suchform wählen" ab Zeile 3710 wird der Index der gewünschten Suchform in das entsprechende Such-Flag eingelesen. Mit Hilfe der geretteten Cursor-Zeile (d6) wird dann in Zeile 2260 das entsprechende Suchformzeichen vor die Feldbezeichnung gesetzt und zur Abfrage nach weiteren Suchfeldern zurückgesprungen.

In den Zeilen 2262-2267 wird die gewünschte Anzahl der Übereinstimmungen in "ue" geschrieben. Über die Zeilen 2280 - 2360 werden die Such-Strings in das Feld s$(f) eingelesen. Da die eigentliche Suchroutine in jedem Datensatz die Felder von einem Startfeld bis zu einem Endfeld untersucht, wird in Zeile

2372-2378 der niedrigste und höchste Feldindex festgestellt und in "fb", "fe" gerettet, um so die Suchzeit zu optimieren.

Die Hauptsuchschleife über die einzelnen Datensätze beginnt ab Zeile 2380. In Zeile 2390 werden die zu untersuchenden Strings in das Feld "u$(i)" übertragen. In der Zeile 2410 wird der Index für gefundene Übereinstimmungen "n" auf null und "f" auf das Startfeld gesetzt, bevor in die eigentliche Suchroutine ab Zeile 100 gesprungen wird.

Die Suchroutine liegt aus Zeitgründen am Programmanfang, da die Zieladresse nach einem Sprung (GOTO, GOSUB) entweder hinter der aufrufenden Zeile oder ab Programmanfang gesucht wird. Würde die Routine am Programmende liegen, müßte bei einem Sprung in eine vorherige Zeile immer das ganze Programm nach der Zielzeilennummer durchsucht werden, was bei längeren Dateien die Suchzeit merklich verlängern würde. Am Anfang der Suchroutine wird getestet, ob schon alle gewünschten Felder durchsucht wurden oder ob die gewünschte Anzahl der Übereinstimmungen gefunden wurde. Ist das nicht der Fall, wird über "sf(f)" der gewünschte Vergleich angesprungen.

Ist der entsprechende Vergleich logisch WAHR, wird der Übereinstimmungszähler erhöht. Nachdem der Feldindex "f" erhöht wurde, wird zum Start zurückgesprungen. Nach dem Rücksprung in die Zeile 2410 wird, wenn die gewünschte Übereinstimmungszahl erreicht wurde, die Datensatznummer in das Feld "gf%()" geschrieben und die Trefferzahl "z" inkrementiert. Nach Abschluß des Suchens werden die Treffer analog zum Unterprogramm "Blättern" angezeigt (2430-2550).

Liste ausgeben (2690 - 2970)

Diese Routine gibt die durch das Feld "gf%(i)" bestimmten Datensätze entweder auf den Bildschirm oder dem Drucker aus. Die Abfrage nach den auszudruckenden Feldern ist analog zu der Feldbestimmung im Unterprogramm "Suchen".

Sortieren (2980 - 3510)

Nach der Abfrage, ob eine Liste erstellt wurde, und nach der Ausgabe der Überschrift auf den Bildschirm (2990-3000) wird das Feld, nach dem sortiert werden soll, abgefragt, und der entsprechende Index in "sf" geschrieben. Für die folgende Sortierroutine werden einige Hilfsfelder benötigt, deren Größe vom Listenumfang abhängt. Da Felder aber normalerweise nur einmal dimensioniert werden dürfen, hätte ich bei der Initialisierung die Hilfsfelder "n$(i)" und "(i)" mit der maximalen Datensatzanzahl dimensionieren müssen. Das würde einerseits viel Speicherplatz schlucken und andererseits die Lade- und Speicherzeit vom bzw. auf Band verlängern. Mit einem Kunstgriff ist es möglich, Fehler selektiv zu löschen und somit redimensionierbar zu machen.

Dieser Kunstgriff besteht darin, vor einer neuen Dimensionierung die Ober- und Untergrenze des Bereichs, in dem der Computer die Felder ablegt zu retten. Die danach dimensionierten Felder können dann durch Rücksetzen der Zeiger auf die alte Länge der Feldertabelle gelöscht werden.

Aus diesem Grunde werden in den Zeilen 3040 und 3050 die Zeiger auf den Start und das Ende der Feldertabelle in "fl" und "fh" gerettet. In der Zeile 3060 werden die Hilfsfelder entsprechend des Listenumfangs dimensioniert. Durch die Schleife in den Zeilen 3070 bis 3090 wird das entsprechende Datensatzfeld eines jeden in der Liste befindlichen Datensatzes in "n$(i)" übertragen. Dieses Stringfeld wird später sortiert. Damit danach die entsprechenden anderen Felder eines Datensatzes dem sortierten Feld zugeordnet werden können, wird in "(i)" ein "Pointer" angelegt, der die Position des entsprechenden Datensatzes innerhalb der Liste enthält.

Die Sortierroutine verarbeitet nur Felder mit Indizes ab eins. Im Datenfeld "e$(f,r)" sind die Datenfelder aber ab Index null abgespeichert. Aus diesem Grund müssen die Daten und der Pointer jeweils in eine Feldvariable mit einem Index, der um eins höher ist als der im Datenfeld, umgespeichert werden. Bei der Umspeicherroutine wird der erhöhte Index dreimal benötigt (Index von "n$", Index von "z", Pointer in "(i)"). Damit innerhalb

der Umspeicherschleife nicht dreimal "i+1" berechnet werden muß, habe ich aus Zeitgründen eine Hilfsvariable "q" definiert, die innerhalb der Schleife nur einmal inkrementiert werden muß.

In Zeile 3100 wird in die Sortierroutine (3240-3510) verzweigt. Diese Routine sortiert das eindimensionale Feld "n$(i)" durch entsprechendes Vertauschen der Inhalte von "n$(i)". Bei jedem Tauschvorgang wird auch der Inhalt des Pointers vertauscht. Am besten wird dies durch ein Beispiel klar. Nehmen wir an, fünf Namen sollen sortiert werden, die in folgender Reihenfolge in den Feldern stehen:

```
n$(1) = Max      (1) = 1
n$(2) = Alfred   (2) = 2
n$(3) = Peter    (3) = 3
n$(4) = Dirk     (4) = 4
```

Nach dem Sortieren erhalten wir:

```
n$(1) = Alfred   (1) = 2
n$(2) = Dirk     (2) = 4
n$(3) = Max      (3) = 1
n$(4) = Peter    (4) = 3
```

Der Index von "z" gibt nun die aufsteigende Reihenfolge der sortierten Strings an und der Inhalt von jedem "z" die Datensatznummer, die an der entsprechenden Position stehen muß. In der Schleife Zeile 3110-3200 werden die Datenfelder entsprechend dem Pointer vertauscht.

Die äußere Schleife zählt über die Felder. In der ersten inneren Schleife wird das entsprechende Feld (durch x bestimmt) aller Datensätze in der alten Reihenfolge in "n$(i)" übertragen.

In der zweiten inneren Schleife werden dann die Inhalte von "n$(i)" entsprechend dem Pointer "z(i)" in das Datenfeld sortiert zurückgeschrieben. Ab Zeile 3210 werden die Hilfsfelder wieder gelöscht. Zuerst werden die Strings gelöscht, um den String-Vektor im Merkheft des Rechners zurückzusetzen. Über das Unterprogramm in den Zeilen 3520 - 3540 wird zu dem aktuel-

len Start der Feldertabelle die gerettete Länge addiert und als Feldertabellenende in den Vektor 49, 50 geschrieben.

Ich hoffe, daß Ihnen nun die Arbeitsweise dieses Programms einigermaßen klar geworden ist und Sie nun wissen, wo und wie Sie dieses Programm Ihren eigenen Wünschen noch besser anpassen können. Ich könnte mir z.B. noch folgende Anpassungen denken:

- Druckerausgabe der Datei entsprechend formatierter Ausgabe.
- Help-Seite, diese Bildschirmausgabe könnte man auf einer separaten Bildschirmseite schreiben, die dann mit Hilfe von POKE-Befehlen mit der aktuellen Bildschirmseite ausgetauscht wird. Das ist eine Anpassung, die besonders interessant ist. Hier könnte man die zusätzliche Bildschirmseite unter das Betriebssystem-ROM (ab $A000) schreiben, so daß kein BASIC-Speicherplatz verlorengeht.

10.12 CC-Inhalt und Katalog für FastTape-Kassetten

Weiter vorne habe ich Ihnen ein Programm beschrieben, das ein Inhaltsverzeichnis Ihrer Kassetten erstellt. Solch eine Inhaltsübersicht möchten Sie sich bestimmt auch für Ihre FastTape-Kassetten anlegen. Darum finden Sie hier ein Listing eines Programms, welches genauso arbeitet.

```
PROGRAMM:FASTT.INHALT

90  REM ************************************
95  REM *                                  *
100  REM *    INHALTSVERZEICHNIS            *
101  REM *                                  *
103  REM *  VON FASTTAPE CASSETTEN          *
104  REM *                                  *
105  REM *    (C) DIRK PAULISSEN            *
106  REM *                                  *
107  REM ************************************
108  REM  V=.7: KL=.54 : REM KONSTANTEN FUER ZAEHLERBERECHNUNGEN C60
109  V = .5: KL = .482:  REM  KONSTANTEN FUER ZAEHLERBERECHNUNGEN C90
110  PA = 828:  REM  STARTADRESSE IM CASSETTENPUFFER
115  SA = 171 :  REM  SEKUNDAERADRESSE +1
116  PRINT "<CLR HOME>< 11 SPACE>INHALTSVERZEICHNIS"
```

```
117  PRINT :  PRINT "< 10 SPACE>FUER FASTTAPE CASSETTEN"
120  PRINT "< 2 CRSR DOWN>DRUCKEN ? <COMM :>J/N<COMM ;>< 2 SPACE>";D$
130  GET D$:  IF D$ = "" THEN 130
136 TA$ =  CHR$ (16)
140 T1$ = TA$ + "05": T2$ = TA$ + "10": T3$ = TA$ + "19": T4$ = TA$ +
"27": T5$ = TA$ + "35"
142 T6$ = TA$ + "43": T7$ = TA$ + "61":  REM  TAB'S
145  IF D$ <  > "J" THEN 190
150  OPEN 2,4 :  REM  DRUCKER OEFFNEN
160  PRINT# 2,T1$"LFN";T2$"ZAEHLER";T3$"K-BYTE";T4$" ANFANG"T5$"< 3
SPACE>ENDE";
162  PRINT# 2,T6$"< 5 SPACE>NAME"T7$"SEKADR."
165  PRINT# 2
190  SYS 49674
200 A$ = ": " +  RIGHT$ ("< 5 SPACE>" +  STR$ ( PEEK (PA) + (256 *
PEEK (PA + 1)) - 1),6)
210 B$ = "C" +  RIGHT$ ("< 4 SPACE>" +  STR$ ( PEEK (PA + 2) + (256 *
PEEK (PA + 3)) - 1),6)
220 C$ = ":  ":  FOR I = 5 TO 20: C$ = C$ +  CHR$ ( PEEK (PA + I)):
NEXT : C$ = C$ + ": "
230 T =  VAL ( RIGHT$ (B$,5)) -  VAL ( RIGHT$ (A$,5)): K$ =  STR$ ( INT
(T / 1024 * 1000 + .5) / 1000)
235 K$ =  RIGHT$ ("< 7 SPACE>" +  STR$ ( INT (T / 1024 * 1000 + .5) /
1000),6)
240 SA$ =  STR$ ( PEEK (SA) - 1)
250  PRINT Z;K$A$B$:  PRINT C$:  PRINT
252 FA = 1 + 2 * <PI> * (Z / 4400):  REM  ABHAENGIGER FAKTOR ZUR
ZAEHLERBERECHNUNG
255 Z1 = Z: Z =  INT (Z + V * FA + (T / 1024) * (KL * FA)):  REM
BANDZAEHLER BERECHNEN
256  PRINT "NACHSTER ZAEHLERSTAND CA."Z
260 N = N + 1: N$ =  RIGHT$ ("< 6 SPACE>" +  STR$ (N),3)
270  IF D$ = "J" THEN  PRINT#
2,T1$;N$;T2$;Z1;T3$;K$;T4$;A$;T5$;B$;T6$;C$;T7$SA$
300  GOTO 190
```

Das in Unterkapitel 10.8 beschriebene Katalogprogramm arbeitet nach entsprechender Änderung ebenfalls mit FastTape. Sie müssen nur die normalen LOAD-Befehle gegen FastTape-Befehle austauschen. Da die Synchronisation bei FastTape-Programmen bedeutend kürzer ist als bei normal abgespeicherten Programmen, empfiehlt es sich, die Programme nicht direkt hintereinander abzuspeichern, sondern jeweils ca. 5 sec Zwischenraum zu lassen.

10.13 Backup von CC auf Disk

Dieses Unterkapitel ist für diejenigen bestimmt, die sowohl ein Diskettengerät besitzen als auch einen Datenrekorder. Eine Datenkassette ist bedeutend robuster und weniger störungsanfällig als eine Diskette. Darum lohnt es sich auch für einen Diskettenbesitzer, wichtige Programme auf Kassette zu speichern. Auch gibt es viele, die selbst nur einen Kassettenrekorder haben, aber einen Freund besitzen, der eine Diskettenstation hat. Mit den folgenden Programmen ist es nun viel einfacher, Programme untereinander auszutauschen.

10.13.1 Backup von Kassette auf Disk

Mit diesem Backup-Programm haben Sie die Möglichkeit, mehrere FastTape-Programme, die hintereinander auf einer Kassette gespeichert sind, auf eine Diskette umzuspeichern.

Für das Programm ist es unerheblich, ob es sich um BASIC- oder Maschinenspracheprogramme handelt. Die Programme werden alle so kopiert, daß sie voll lauffähig sind, egal, von welchem Speichermedium Sie es wieder einladen. Wichtig ist nur, daß alle Programme, die Sie kopieren wollen, einen Namen haben, da die Diskettenstation im Gegensatz zum Datenrekorder namenlose Files nicht akzeptiert. Da das Backup-Programm mit FastTape arbeitet, müssen Sie das FastTape-Programm laden, bevor Sie das Backup-Programm starten. Die Bedienung ist denkbar einfach.

Geben Sie das Programm in Ihren Rechner ein und speichern Sie es vor dem Starten ab, da es sich selbst nach dem Starten löscht. Wenn Sie nun das Programm starten, wird zuerst das Maschinenprogramm in den Speicher ab $C3C0 hinter das FastTape-Programm geschrieben. Dann werden Sie nach der Anzahl der zu kopierenden Programme gefragt. Dieser Wert wird in der Speicherstelle 767 für das Maschinenprogramm zwischengespeichert.

Als nächstes müssen Sie eine formatierte Diskette in das Diskettenlaufwerk und die Kassette mit den zu kopierenden Program-

men in den Datenrekorder legen. Nach Drücken der <Return>-Taste verzweigt das BASIC-Programm in das Maschinenprogramm. Dies liest nun das erste Programm in den BASIC-RAM (der komplette BASIC-RAM wird als Puffer benutzt). Danach wird das Programm aus dem Speicher auf Diskette geschrieben. Dieser Vorgang wird nun der Anzahl der Programme entsprechend wiederholt. Am Ende kehrt der Rechner wieder in den Direktmodus zurück. Wollen Sie nun noch weitere Programme kopieren, müssen Sie die Programmanzahl in die Speicherstelle 767 poken und das Maschinenprogramm erneut durch SYS 50112 starten.

```
85 .PAG 61,5
86 .TIT "BACKUP T=>D"
90 .OPT P
100 *= $C3C0
110 :
115 ; ***********************************
120 ; *                                 *
130 ; *   BACKUP VON CC AUF DISKETTE    *
135 ; *                                 *
140 ; *          MIT FASTTAPE           *
145 ; *                                 *
146 ; ***********************************
150 : :  :
160 :
200 CR = 13 ;
205 ABSFLG = 2
210 STARTV = $AC
220 ENDVEC = $AE
230 CASPUF = $33C
240 ANZAHL = $2FF
245 CSTART = $803 ; PUFFERSTART
250 LOADR = $C18E ; FASTTAPE LADEN
260 NEW = $A642 ; BASIC-BEF. NEW
270 ERROR = $A437 ; BASIC-WARMSTART
280 MSSG = $F693 ; MELDUNG AUSGEBEN
290 :
300 STATUS = $FFB7
310 SETLFS = $FFBA
320 SETNAM = $FFBD
325 OPEN = $FFC0
330 CLOSE = $FFC3
335 CHKOUT = $FFC9
340 CLRCH  = $FFCC
350 BSOUT  = $FFD2
950 :
955 :
960 ; HAUPTPROGRAMM
```

```
970 :
1000 START LDA #$00
1010 JSR NEW
1020 JSR CLOAD
1030 JSR DSAVE
1040 DEC ANZAHL
1050 BEQ ENDE
1060 JMP START
1070 ENDE RTS
1082 :
1084 :
1086 ; AUF DISK SPEICHERN
1088 :
1089 DSAVE LDA #$10
1090 LDX #$41 ; NAMENPARAMETER
1092 LDY #$03 ; SETZEN
1093 JSR SETNAM
1095 JSR MSSG ; ' SAVING NAME ' AUSGEBEN
1100 LDA #CR
1110 JSR BSOUT ; NEUE ZEILE
1120 LDX #$10
1130 ELOOP DEX
1140 LDA CASPUF+5,X
1150 CMP #$20
1160 BNE ENDNAM
1170 CPX #$00
1180 BNE ELOOP
1190 ENDNAM INX ; ENDE DES FILE-NAMENS
1200 LDA #"," ; SUCHEN UND ' ,P,W '
1210 STA CASPUF+5,X
1220 INX ; ANHAENGEN
1230 LDA #"P"
1240 STA CASPUF+5,X
1250 INX
1260 LDA #","
1270 STA CASPUF+5,X
1280 INX
1290 LDA #"W"
1300 STA CASPUF+5,X
1310 INX
1320 TXA ; LAENGE DES FILE-NAMENS
1330 LDX #$41 ; STARTADR DES    ' '
1340 LDY #$03
1350 JSR SETNAM
1360 LDA #$08 ; FILE-NUMMER
1370 LDX #$08 ; GERAETENUMMER
1380 LDY #$01 ; SECUNDAERADR.
1390 JSR SETLFS
1400 JSR OPEN
1410 LDX #$08
1420 JSR CHKOUT
1430 LDA #<CSTART
1440 LDX #>CSTART
```

```
1450 STA STARTV
1460 STX STARTV+1
1470 LDA CASPUF+2 ; ENDADRESSE BERECHNEN
1480 SEC ; AUS DIFFERENZ START- + ENDADR.
1490 SBC CASPUF ; AUS DEM HEADER PLUS BESTIMMTER
1500 PHP ; STARTADERSSE
1510 CLC
1520 ADC STARTV
1530 STA ENDVEC
1540 LDA CASPUF+3
1550 ADC STARTV+1
1560 PLP
1570 SBC CASPUF+1
1580 STA ENDVEC+1
1590 LDA CASPUF ; PROGRAMMSTARTADR.
1600 JSR BSOUT ; ZUR DISK SENDEN
1610 LDA CASPUF+1
1620 JSR BSOUT
1622 :
1623 ; SPEICHERSCHLEIFE
1624 :
1630 PSAVE LDY #$00
1640 LDA (STARTV),Y
1650 JSR BSOUT
1660 INC STARTV ; ADRESSE ERHOEHEN
1670 BNE NOTHI
1680 INC STARTV+1
1690 NOTHI LDA STARTV
1700 CMP ENDVEC ; ENDADRESSE ERREICHT
1710 LDA STARTV+1
1720 SBC ENDVEC+1
1730 BCC PSAVE
1740 JSR CLRCH
1750 LDA #$08
1760 JMP CLOSE
1762 :
1764 :
1766 ; VON CC LADEN
1768 :
1770 CLOAD LDX #$00
1780 LDY #<CSTART
1790 LDA #>CSTART
1800 STX $0A ; LOAD/VERIFY FLAG
1810 STX $93
1820 STY $AC
1830 STA $AD
1840 LDA #1+4 ; AN BESTIMMTE ADRESSE,
1850 STA ABSFLG ; NICHT WARTEN
1860 LDA #$00 ; KEINEN FILE-NAME
1870 JSR SETNAM
1880 LDX #$01 ; GA
1890 LDY #$00 ; SA
1900 JSR SETLFS
```

```
1910 JSR LOADR
1922 :
1924 ; STATUS TESTEN
1926 :
1930 HOLSTA JSR STATUS
1940 AND #$BF
1950 BEQ OK
1960 LDX #$1D
1970 JMP ERROR
1980 OK RTS
1990 .END
```

PROGRAMM:BACKUP-CC-DISK.B

```
100   GOSUB 240
110   PRINT "<CLR HOME>*******************************************<CRSR
LEFT><CRSR DOWN>*<CRSR UP>*"
120   PRINT "* B A C K U P< 3 SPACE>M I T< 2 SPACE>F A S T T A P E **< 2
CRSR LEFT><CRSR DOWN>*";
130   PRINT "*< 8 SPACE>VON CASSETTE AUF DISKETTE< 5 SPACE>**< 2 CRSR
LEFT><CRSR DOWN>*";
140   PRINT "*< 11 SPACE>(C) DIRK PAULISSEN< 9 SPACE>**< 2 CRSR
LEFT><CRSR DOWN>*";
150   PRINT "*******************************************"
160   PRINT :  PRINT :  PRINT "WIEVIELE PROGRAMME WOLLEN SIE"
170   PRINT :  INPUT "COPIEREN ";AZ
180   POKE 767,AZ
190   PRINT :  PRINT :  PRINT "LEGEN SIE DIE QELLCASSETTE EIN"
200   PRINT "<CRSR DOWN>UND DRUECKEN SIE << RETURN >>"
210   GET A$:  IF A$ < > CHR$ (13) THEN 210
220   SYS 50112
230   END
240   REM  LADEPROGRAMM  FASTCOPY T-D 64
250  E = 50333: A = 50112: PS = 0
260   FOR I = A TO E:  READ X:  POKE I,X: PS = PS + X:  NEXT
270   IF PS < > 27517 THEN  PRINT "FEHLER IN DATAS":  END
280   RETURN
290   DATA 169,0,32,66,166,32,112,196,32,212,195,206,255,2,240,3,76,192,
195,96
300   DATA 169,16,162,65,160,3,32,189,255,32,147,246,169,13,32,210,255,1
62,16
310   DATA 202,189,65,3,201,32,208,4,224,0,208,244,232,169,44,157,65,3,2
32,169
320   DATA 80,157,65,3,232,169,44,157,65,3,232,169,87,157,65,3,232,138,1
62,65
330   DATA 160,3,32,189,255,169,8,162,8,160,1,32,186,255,32,192,255,162,
8,32,201
340   DATA 255,169,3,162,8,133,172,134,173,173,62,3,56,237,60,3,8,24,101
,172,133
350   DATA 174,173,63,3,101,173,40,237,61,3,133,175,173,60,3,32,210,255,
173,61
360   DATA 3,32,210,255,160,0,177,172,32,210,255,230,172,208,2,230,173,1
65,172
```

```
370  DATA 197,174,165,173,229,175,144,233,32,204,255,169,8,76,195,255,1
62,0,160
380  DATA 3,169,8,134,10,134,147,132,172,133,173,169,5,133,2,169,0,32,1
89,255
390  DATA 162,1,160,0,32,186,255,32,142,193,32,183,255,41,191,240,5,162
,29,76
400  DATA 55,164,96
```

10.13.2 Programmbeschreibung Backup CC-DISK

Für diejenigen Leser, die zumindest Grundkenntnisse in Assembler-Programmierung haben, möchte ich das Assembler-Listing etwas näher erklären. Die Hauptschleife des Programms steht ab $C30C0 bis $C3D3. Zuerst werden durch den NEW-Befehl alle BASIC-Vektoren zurückgesetzt. Dann wird zur Kassetten-Load-Routine CLOAD verzweigt; nach der Ausführung dieses Unterprogramms wird das Programm durch DSAVE auf Disk gespeichert. Nachdem ANZAHL dekrementiert und getestet wurde, wird zum Start zurückgesprungen.

```
CLOAD ($C469)
```

Der Pufferstart CSTART wird in die Speicherstellen $AC, $AD übertragen. Aus diesen Speicherstellen holt sich FastTape die erforderliche Ladeadresse. Weiterhin werden die LOAD/VERIFY-Flags auf 0 = LOAD gesetzt. Durch das Setzen von Bit 0 und 2 in ABSFLG wird das FastTape-Programm veranlaßt, an die Adresse zu laden, die in $AC und $AD übergeben wird, und nach dem Finden des Programms nicht zu stoppen. Ab $C47B werden die notwendigen Vorbereitungsroutinen des Betriebssystems SETNAM und SETLFS aufgerufen. Danach wird zum FastTape-Unterprogramm LOADR verzweigt. Nach dem Ladevorgang wird der STATUS getestet und in die Hauptschleife zurückgesprungen.

DSAVE

Das Speichern auf Diskette ist etwas komplizierter. Zuerst wird über Betriebssystem-Routinen "SAVING name" ausgegeben. Von $C3DE bis $C3EA wird das Ende des File-Namens, der im

Kassettenpuffer steht, festgestellt, indem vom Ende her so lange auf <SPACE> getestet wird, bis ein anderes Zeichen gefunden wird.

In den folgenden Zeilen bis $C401 wird ",P,W" (Programm, Write) an den File-Namen angehängt. Über das X-Register wird die Länge des kompletten File-Namens bestimmt. Ab $C465 beginnen die notwendigen Vorbereitungsroutinen zur Diskettenspeicherung SETNAM, SETLFS, OPEN, CHKOUT. Danach wird der Startvektor wieder auf den Pufferstart gesetzt. Die Endadresse des eingeladenen Programms wird aus der Differenz von Start und Endadresse aus dem Kassettenpuffer plus Startadresse des Puffers berechnet ($C426 - $C43C). Bei der Diskettenspeicherung geben die ersten beiden Bytes die Adresse an, von der es abgespeichert wurde. Diese Adresse steht als Startadresse im Kassettenpuffer. Sie wird in $CH3E - $C447 an die Diskette übermittelt.

Ab PSAVE ($C44A) beginnt die eigentliche Programmspeicherung. In das Y-Register wird 0 geladen, und die Programm-Bytes werden "indirekt" indiziert in den Akkumulator geladen und über BASOUT an die Diskettenstation gesandt. Danach wird der Startvektor inkrementiert und mit dem Endvektor verglichen. Sind sie nicht gleich, wird wieder nach PSAVE gesprungen. Sind die Vektoren gleich, wird der Diskettenausgabekanal geschlossen und zur Hauptschleife zurückgesprungen.

10.13.3 Backup von Diskette auf Kassette

Mit diesem Programm haben Sie die Möglichkeit, sehr komfortabel ein oder mehrere Programme von einer Diskette auf eine oder mehrere Kassetten zu Übertragen. Die Eingabe erfolgt über einen BASIC-Lader, den Sie am Anschluß an die Programmbeschreibung finden. Vergessen Sie auch bei diesem BASIC-Lader nicht, ihn vor dem Starten abzuspeichern. Dieses Programm läuft ebenfalls nur, wenn Sie FastTape schon geladen haben.

Bedienung des Programms

Gestartet wird das Programm durch SYS 50176. Sofort erscheint der Titel auf dem Bildschirm, und Sie werden gebeten, die Quelldiskette einzulegen und eine Taste zu drücken.

Nach kurzer Zeit erscheint der Diskettentitel und der erste File-Eintrag des Disketteninhaltsverzeichnisses mit der Frage auf "Ja/Nein". Wenn Sie das Programm kopieren wollen, drücken Sie "J", und es erscheint JA hinter dem File-Eintrag. Dann wird der nächste Eintrag angezeigt. Wollen Sie ein Programm nicht kopieren, drücken Sie "N", und der Eintrag wird mit NEIN gekennzeichnet. Auf diese Weise können Sie bis zu 48 Programme mit JA kennzeichnen.

Falls Sie sich innerhalb des Programms einmal vertippen, können Sie durch Drücken der <Run/Stop>-Taste immer das Programm abbrechen und zur Titelseite zurückkehren. Wenn Sie das ganze Inhaltsverzeichnis bearbeitet haben, werden Sie gefragt, ob Sie die Programme einzeln oder durchgehend kopieren wollen. Möchten Sie alle Programme auf einer Diskette speichern, geben sie "1" ein. Wenn Sie eine "2" eingeben, sagt Ihnen der Computer immer, welches Programm er als nächstes einliest und wartet auf einen Tastendruck. Damit ist es möglich, für jedes Programm eine andere Kassette einzulegen. Am Ende des Kopiervorgangs kehrt der Rechner wieder zur Titelseite zurück. Sie haben dann die Möglichkeit, eine neue Diskette zu kopieren oder durch Drücken von "E" ins BASIC zurückzukehren.

10.13.4 Programmbeschreibung Backup Disk-CC

Für Interessierte und der Assembler-Sprache Kundige nun noch eine Beschreibung des nicht gerade kurzen Programms.

Vorbemerkung

Dieses Programm arbeitet mit zwei unterschiedlichen Lade- bzw. Speicherstartadressen. Die tatsächliche Startadresse ist die Adresse des Puffers, unter der das Programm zum kopieren

zwischengespeichert wird. Die eigentliche Startadresse ist die, unter der das Programm normalerweise steht und arbeitet.

Das Programmprinzip ist folgendes

Nach der Disketten-Initialisierung werden zuerst die File-Namen eingelesen und im Kassettenpuffer ab $0340 zwischengespeichert. Dann wird der Anwender gefragt, ob das Programm kopiert werden soll oder nicht. Wird die Frage mit "J" beantwortet, wird getestet, ob die maximale Blocklänge bzw. die File-Namen-Listenlänge überschritten wird und ggf. eine Fehlermeldung ausgegeben. Wird keine Überschreitung festgestellt, werden der File-Name und seine Länge in zwei Tabellen (NAMTAB, LNGTAB) hinter dem Backup-Programm übertragen. Weiterhin testet das Programm auch auf den File-Typ des zu kopierenden Files. Handelt es sich nicht um ein Programm-File, wird eine Fehlermeldung ausgelesen.

Wurde auf diese Weise das ganze Inhaltsverzeichnis bearbeitet, wird in die eigentliche Kopierroutine gesprungen. Nach Ausgabe von "READING name" wird der entsprechende File-Name wieder in den Kassettenpuffer übertragen und ",P,R" angehängt. Mit dem so erweiterten File-Namen wird eine Lesedatei eröffnet und das entsprechende Programm nach $0900, der tatsächlichen Programmlade-Adresse (TLSADR), geladen. Wenn beim Lesen kein Fehler aufgetreten ist, wird der FastTape-Speicherroutine die Programmstart- und endeadresse, die in den Programm-Header geschrieben werden soll, und die tatsächliche Programmstart- und endeadresse, unter der das Programm zur Zeit abgespeichert wurde, übergeben. Danach wird zur FastTape-Routine "ABSOLUT" verzweigt. Anschließend wird wieder zum Start zurückgesprungen, falls noch weitere Files kopiert werden sollen.

```
60 .OPT P
95 ;*************************************
96 ;*                                   *
100 ;*             BACKUP               *
110 ;*                                  *
115 ;*   VON DISKETTE AUF CASSETTE      *
117 ;*                                  *
120 ;*          MIT FASTTAPE            *
```

```
122 ;*                                  *
125 ;************************************
130 ;
132 ; START MIT SYS 50176
140 ;
142 *= $7400 ;
145 :START = *
150 CLSCRN = $E55F; BILDSCHIRM LOESCHEN
152 NUMOUT = $DDCD ; POS. INEGERZAHL AUSGEB.
154 MOTAUS = $FD08 ; MOTOR AUSSCHALTEN
156 PTASTE  = $F8B7 ; RECORDERTASTE ABFRAGEN
158 SFINUM = $F3CF ; SUCHT FILE-NUMMER
160 STROUT = $CB1E ; STRING AUSGEBEN
162 SETPAR = $F3DF ; ZT FILE-PARAMETER
164 ABSOLUT = $707F  ; EINSPRUNG IN FASTTAPE
169 : :
170 GETBYT = $FFE4 ;BYT VOM EINGABEGERAET HOHLEN
180 SETLFS = $FFBA ;FILE-PARAMETER SETZEN
190 SETNAM = $FFBD ;FILE-NAME-PARAMETER SETZEN
200 OPEN   = $FFC0
210 CLOSE = $FFC3
220 TALK   = $FFB4
230 SETALK = $FF96 ;SECUNDAERADR. NACH TALK
240 IECIN = $FFA5 ;BYTE VOM IEC-BUS HOLEN
250 CLALL = $FFE7
260 UNTALK = $FFAB ;UNTALK AUSGEBEN
280 STOP   = $FFE1 ;<STOP>-TASTE ABFRAGEN
290 BASOUT = $FFD2 ;BYT AUSGEBEN
300 : :
320 GA     = $BA ;GERAETEADRESSE
330 SA     = $B9 ;SEKUNDAERADRESSE
340 MAXBLK = $5D ;MAX. LADBARE BLOECKE
350 CR     = $0D ;CARRIDGE RETURN
360 COPANZ = $3 ;AKTUELLE LISTENLAENGE
370 TEMP = $4 ;HILFSPUFFER
380 FLAG   = $5 ;EINZEL-FLAG
390 LISTPT = $41 ;POINTER AUF NAMENLISTE
400 FILPUF = $340 ;FILE-PUFFER
410 LNGTAB = START+$640 ;TAB. DER NAMENLAENGEN
420 NAMTAB = START+$700 ;TAB. DER FILE-NAMEN
430 LSTMAX = $30 ;MAXIMALE LISTENLAENGE
440 TLSADR = $13 ;HI-BYTE DER LADEADRESSE
450 TEMP2  = $22      ;HILFSPOINTER
460 TPGSTA = $AC      ;PTR ZUM TATS. PRGSTART
470 TPGEND = $AE      ;PTR ZUM TATS. PRGENDE
480 PRGSTA = $A7      ;PTR ZUR EIGENTL. PRGSTART
490 PRGEND = $A9      ;PTR ZUM EIGENTL. PRGENDE
495 MOFLAG = $C0      ;MOTOR-FLAG
520 ;
530 ;
560 LDA #25
570 STA $900F
580 ;STA $D021
```

```
590  LDA #6
600  STA $286
610    LDA #<TITEL
620  LDY #>TITEL
630  JSR STROUT
640 LDA #<QUEINS ;"QUELLDISK EINSTECKEN "
650        LDY #>QUEINS
660        JSR STROUT
670  JSR WAIT
680 CMP #"E" ;PROGRAMM BEENDEN
690        BNE OKCOPY
700        RTS
701 :
702 ; INHALT EINLESEN
703 :
710 OKCOPY JSR CARET
720  JSR CARET
730 JSR INIT ;DISK INITIALISIEREN
740  LDA #$08
750  TAX
760  LDY #$00
770  JSR SETLFS
780  LDA #$01
790 LDX #<CATALO
800 LDY #>CATALO
810  JSR SETNAM
820  JSR OPEN
830  LDA #$08
840  JSR TALK
850  LDA #$00
860  JSR SETALK
870  LDY #$04
880 LOOP1   JSR IECIN ;DIE ERSTEN 4 BYTES EINLESEN
890  DEY
900  BNE LOOP1
910 JSR IECIN ;BLOCKZAHL EINLESEN
920  STA TEMP2
930  JSR IECIN
940  LDX TEMP2
950 JSR NUMOUT ;UND AUSGEBEN
960  JSR SPACE
970 LOOP2   JSR IECIN ;DISKNAME ETC AUSGEBEN
980  BEQ NULL
990  JSR BASOUT
1000  CLC
1010  BCC LOOP2
1020 NULL  JSR CARET
1030  JSR CARET
1040  JSR IECIN
1050  JSR IECIN
1060 LDY #$00 ;ANZAHL DER ZU COPIERENDEN
1070 STY COPANZ ;FILES AUF NULL SETZEN
1071 :
```

```
1072 ; INHALT AUSGEBEN UND FRAGEN
1073 :
1080 LSTART  JSR IECIN
1090  STA TEMP
1100  JSR IECIN
1110  STA TEMP+1
1120  LDX TEMP
1130 JSR NUMOUT ;BLOCKZAHL AUSGEBENN
1140  JSR SPACE
1150 LDY #$00 ;FILE-NAMEN
1160 WRFILE  JSR IECIN ;EINLESEN
1170  PHA
1180 JSR AUSGAB ;AUSGEBEN
1190 PLA ;UND IN
1200 STA FILPUF,Y ;PUFFER SCHREIBEN
1210  BEQ NAMEND
1220  INY
1230  BNE WRFILE
1240 NAMEND  JSR IECIN
1250  JSR IECIN
1260 LDA $90 ;STATUS ABFRAGEN
1270  BEQ FRAGEN
1280  JMP INHEND
1290 FRAGEN  LDA TEMP+1
1300         BNE TOLONG
1310  LDA TEMP
1320 CMP #MAXBLK ;AUF MAXIMALE LAENGE
1330 BCC LNGEOK ;TESTEN
1340 TOLONG LDA #<SLONG ;"ZU LANG ZUM
1350 LDY #>SLONG ;ZUM COPIEREN"
1360 JSR STROUT ;AUSGEBEN
1370 FINISH  JMP NXFILE
1380 LNGEOK  LDA COPANZ
1390         CMP #LSTMAX
1400  BCC FANZOK
1410 LDA #<LILONG ;"LISTE ZU LANGE"
1420         LDY #>LILONG
1430 JSR STROUT ;AUSGEBEN
1440  CLC
1450  BCC FINISH
1460 FANZOK LDA #$00 ;"JA/NEIN" AUSGEBEN
1470  STA $08
1480  LDA #$1F
1490  STA $D3
1500         LDA #<JANEIN
1510         LDY #>JANEIN
1520  JSR STROUT
1530 LOOP3  JSR INPUT
1540         CMP #"N"
1550  BEQ NEIN
1560         CMP #"J"
1570  BNE LOOP3
1580 LDA #<STRJA ;JA AUSGEBEN
```

```
1590         LDY #>STRJA
1600  JSR STROUT
1610  LDA COPANZ
1620  JSR SETPTR
1630  LDX #$00
1640 LOOP4  INX ;FILE-NAMEN-START SUCHEN
1650  LDA FILPUF,X
1660 CMP #$22 ;1.HOCHKOMMA = START
1670  BNE LOOP4
1680 STX TEMP+1 ;START RETTEN
1690  INX
1700 UMSETZ LDA FILPUF,X ;NAMEN IN LISTE SCHREIBEN
1710 CMP #$22 ;2. HOCHKOMMA = ENDE
1720  BEQ UMSEND
1730  STA (LISTPT),Y
1740  INX
1750  INY
1760  BNE UMSETZ
1770 UMSEND TXA ;ENDPUNKT IN AKKU
1780 LDY COPANZ ;INDEX IN Y-REG.
1790  CLC
1800 SBC TEMP+1 ;LAENGA BERECHNEN
1810 STA LNGTAB,Y ;IN TABELLE SCHREIBEN
1820 WETEST  LDA FILPUF,X
1830  BNE TYPTST
1840 ILL     LDA #<STRILL ;"ILLEGALER FILE-TYP"
1850         LDY #>STRILL
1860 JSR STROUT ;AUSGEBEN
1870  JMP NXFILE
1880 TYPTST CMP #"S" ;TESTET AUF PRG-FILE
1890         BEQ ILL
1900         CMP #"P"
1910  BEQ TYPOK
1920  INX
1930  BNE WETEST
1940 TYPOK  INC COPANZ ;ANZ DER ZU COPIERENDEN
1950 CLC ;FILES ERHOEHEN
1960  BCC NXFILE
1970 NEIN   LDA #<STRNO ;"NEIN" AUSGEBEN
1980         LDY #>STRNO
1990  JSR STROUT
2000 NXFILE  JSR CARET
2010  JMP LSTART
2020 INHEND LDA #$08 ;FILE SCHLIESSEN
2030  JSR CLOSE
2040 LDA COPANZ ;SOLLEN FILES COPIERT
2050 BNE COPYST ;WERDEN/NEIN=> START
2060         JMP START
2062 : :
2064 ;COPIEREN VORBEREITEN
2066 : :
2070 COPYST LDA #<FRAGE ;"EINZELN ODER
2080 LDY #>FRAGE ; HINTEREINANDER"
```

```
2090 JSR STROUT ;AUSGEBEN
2100 ABFRA2  JSR INPUT
2110        CMP #"1"
2120        BEQ EINZEL
2130        CMP #"2"
2140        BNE ABFRA2
2150 LDA #$00 ;FLAG ENTSPRECHEND
2160  .BYT $2C
2170 EINZEL LDA #$FF ;SETZEN
2180  STA FLAG
2190 LDA #$0F ;FEHLERKANAL SCHLIESSEN
2200  JSR CLOSE
2202 : :
2204 ;COPIER - ROUTINE
2206 : :
2208 ;EINLESEN VON DISK
2209 ;   VORBEREITEN
2210 :LDX #$00 ;TEMP LOESCHEN
2220 STX TEMP ;TEMP => PRG-ZAELER
2230 LOLOOP LDA #<SREAD ;"READING"
2240        LDY #>SREAD
2250 JSR STROUT ; AUSGEBEN
2260 LDY TEMP ;POS. IN TABELLE
2270 LDX LNGTAB,Y ;LAENGE NACH X-REG.
2280  LDA TEMP
2290  JSR SETPTR
2300 WRTNAM LDA (LISTPT),Y ;FILE-NAME AUSGEBEN
2310  JSR BASOUT
2320  INY
2330  DEX
2340  BNE WRTNAM
2350 LDA #$02 ;FILE-NUMMER
2360 LDX #$08 ;GA
2370 TAY ;SA
2380  JSR SETLFS
2390  LDX TEMP
2400  LDA LNGTAB,X
2410  STA TEMP2
2420  LDA TEMP
2430  JSR SETPTR
2440  LDX #$00
2450 LOOP6  LDA (LISTPT),Y ;FILE-NAME IN PUFFER
2460 STA FILPUF,X ;SCHREIBEN
2470  INY
2480  INX
2490  DEC TEMP2
2500        BNE LOOP6
2510  LDY #$00
2520 LOOP7  LDA PRGRE,Y ; ",P,R " ANHAENGEN
2530  STA FILPUF,X
2540  INY
2550  INX
2560  CPY #$04
```

```
2570        BCC LOOP7
2580  TXA
2590        LDX #<FILPUF
2600        LDY #>FILPUF
2610  JSR SETNAM
2620  JSR OPEN
2630        LDA #TLSADR
2640  LDY #$00
2650  STY TEMP2
2660  STA TEMP2+1
2670 ;LDA $D011
2680 ;AND #$EF ;BILDSCHIRM AUS
2690 ; STA $D011
2700 LDA #$0F ;KANAL 15 OEFFNEN
2710 LDX #$08
2720 TAY
2730 JSR SETLFS
2740 LDA #$03 ;"UI-" DISKETTE
2750 LDX #<UIMIN ;SCHNELL SCHALTEN
2760 LDY #>UIMIN
2770 JSR SETNAM
2780 JSR OPEN
2790 LDX #$02 ;LADEKANAL OEFFNEN
2800  JSR SFINUM
2810  JSR SETPAR
2820  LDA GA
2830  JSR TALK
2840  LDA SA
2850  JSR SETALK
2860  LDY #$00
2862 :
2864 ;PROGRAMM IN PUFFER LADEN
2866 :
2870 LDLOOP JSR IECIN ;PRG LADEN
2880  JSR STOBYT
2890  LDX $90
2900  BEQ LDLOOP
2910  JSR FEHLER
2920  PHP
2930  LDA #$02
2940  JSR CLOSE
2950 ;LDA $D011 ; BILDSCHIRM AN
2960 ;ORA #$10 ;
2970 ;STA $D011 ;
2980 PLP
2990  BCC NOFEHL
3000  JSR FEMELD
3002 :
3004 ;AUF CC SPEICHERN VORBEREITEN
3006 :
3010 NOFEHL  LDX TEMP
3020  LDA TEMP2
3030        STA TPGEND
```

```
3040  LDA TEMP2+1
3050        STA TPGEND+1
3060  INX
3070 LDA TEMP
3075 BNE NOMELD
3080 LDA #<ZICASS ;"CASSETTE EINLEGEN "
3090 LDY #>ZICASS
3100 JSR STROUT
3105 JSR INPUT
3110 JSR TASTE ;RECORDERTASTE ABFRAGEN
3115 NOMELD LDA #<WRITI
3120 LDY #>WRITI
3125 JSR STROUT
3130 LDY TEMP
3135 LDX LNGTAB,Y
3140 LDA TEMP
3145 JSR SETPTR
3150 WRNAM2 LDA (LISTPT),Y
3155 JSR BASOUT
3160 INY : DEX
3165 BNE WRNAM2
3210 LDX #$01 ;FILE-PARAMETER
3220 LDA #0 ;FUER CASSETTENSPEI-
3230 TAY ;CHERUNG
3240  JSR SETLFS
3250  LDX TEMP
3260  LDA LNGTAB,X
3270  STA TEMP2
3280  LDA TEMP
3290  JSR SETPTR
3300  LDX #$00
3310 ULOOP  LDA (LISTPT),Y ;FILE-NAMEN IN
3320 STA FILPUF,X ;CASSETTENPUFFER
3330 INX ;SCHREIBEN
3340  INY
3350  DEC TEMP2
3360 BNE ULOOP
3370 TXA ;FILE-NAME-PARAMETER
3380 LDX #<FILPUF ;SETZEN
3390 LDY #>FILPUF
3400  JSR SETNAM
3410  LDY #$00
3420        LDA #TLSADR
3425 STY TPGSTA
3430 STA TPGSTA+1
3440 LDA (TPGSTA),Y ;PRGSTARTADR.
3450 STA PRGSTA ;IN PRGSTARTVEKTOR
3460  INY
3470  LDA (TPGSTA),Y
3480 STA PRGSTA+1
3490  INY
3500  STY TPGSTA
3520 LDA TPGEND ;ENDADRESSE BERECHNEN
```

```
3530  SEC
3540  SBC TPGSTA
3550  PHP
3555 CLC
3560  ADC PRGSTA
3570  STA PRGEND
3580  LDA TPGEND+1
3590  ADC PRGSTA+1
3600  PLP
3610  SBC TPGSTA+1
3620  STA PRGEND+1
3622 :
3624 ;PRG AUF CC SPEICHERN
3626 :
3630 LDX #5 ;SYNCHRONISATIONS
3640 STX $AB ;WIEDERHOLUNG
3650 JSR ABSOLUT ;FASTTAPE SPEICHERN
3690  JSR SCRON
3700  INC TEMP
3710  LDX TEMP
3720  CPX COPANZ
3730  BCS CENDEN
3732 BIT FLAG
3733 BPL EINZE2
3740  JMP NXTFIL
3755 CENDEN LDA #<EOCOP ;"ENDE DES COPIERENS"
3760        LDY #>EOCOP
3770 JSR STROUT ;AUSGEBEN
3780  JSR INPUT
3790        JMP START
3810 EINZE2 JSR NNAMAU
3820 LDA #<ZICASS ;"CASSETTE EINLEGEN "
3830        LDY #>ZICASS
3840 JSR STROUT ;AUSGEBEN
3845 JSR INPUT ;AUF TASTE WARTEN
3850 JSR TASTE
3870 NXTFIL  JSR CARET
3880 JMP LOLOOP ;NAECHSTES FILE LADEN
3882 ::
3884 ;     UNTERPROGRAMME
3886 ::
3887 ;BILDSCHIRM ANSCHALTEN
3888 :
3890 ;SCRON  JSR SEUIP
3900 ; LDA $D011
3910 ; ORA #$10
3920 ; STA $D011
3930 ; RTS
3932 :
3934 ;LISTENCOPYER (LISTPT) AUF
3935 ;AUF ENTSPR. FILE-NAMEN SETZEN
3936 :
3940 SETPTR  LDY #$00
```

```
3950  ASL A
3960  ASL A
3970  STY LISTPT+1
3980  ASL A
3990  ROL LISTPT+1
4000  ASL A
4010  ROL LISTPT+1
4020  STA LISTPT
4030  LDA LISTPT+1
4040  CLC
4050        ADC #>NAMTAB
4060  STA LISTPT+1
4070  LDY #$00
4080  RTS
4082 :
4084 ;BYTE IN DEN SPEICHER SCHREIBEN
4086 :
4090 STOBYT STA (TEMP2),Y
4100 INC TEMP2 ;ZAEHLER ERHOEHEN
4110        BNE NOINC
4120  INC TEMP2+1
4130 NOINC  CLI
4140  RTS
4142 :
4144 ;ASCII-BYTE HOLEN UND IN HEXZAHL
4145 ;         UMWANDELN
4146 :
4150 ASCHEX JSR IECIN
4160 AND #$0F
4170  ASL A
4180  ASL A
4190  ASL A
4200  ASL A
4210  STA $57
4220  JSR IECIN
4230  AND #$0F
4240  ORA $57
4250  RTS
4252 :
4254 ;HEXZAHL IN ASCII-BYTE UMWANDELN
4255 ;        UND AUSGEBEN
4256 :
4260 HEXASC PHA
4270 LSR A
4280  LSR A
4290  LSR A
4300  LSR A
4310        JSR HALBBT
4320  PLA
4330 HALBBT AND #$0F
4340  CLC
4350  ADC #$30
4360 JMP BASOUT ;AUSGEBEN
```

```
4362 :
4364 ;FEHLERMELDUNG AUSGEBEN
4366 :
4370 ;FEMELD  LDA $D011
4380 ;ORA #$10
4390 ; STA $D011
4400 FEMELD LDA #<WMACH ;"WEITER MACHEN ? "
4410         LDY #>WMACH
4420 JSR STROUT ;AUSGEBEN
4430 FLOOP  JSR INPUT ;AUF TASTE WARTEN
4440  CMP #$59
4450 BNE NEIN2
4460  RTS
4470 NEIN2  CMP #$4E
4480  BNE FLOOP
4490  JMP ENDE
4492 :
4494 ;FOLGENDES FILE AUSGEBEN
4496 :
4500 NNAMAU LDA #<SNFILE ;"NAECHSTES FILE "
4510         LDY #>SNFILE
4520 JSR STROUT ;AUSGEBEN
4530 LDA TEMP ;POS IN TABELLE
4540 ASL A ;BERECHNEN
4550  ASL A
4560  ASL A
4570  ASL A
4580  LDX TEMP
4590  LDY LNGTAB,X
4600  TAX
4610 WRNAM3 LDA NAMTAB,X ;FILE-NAMEN AUSGEBEN
4620  JSR BASOUT
4630  INX
4640  DEY
4650  BNE WRNAM3
4660  RTS
4662 :
4664 ;DISKETTE AUF 'LANGSAM' SCHALTEN
4665 ;  UND BILDSCHIRM ANSCHALTEN
4666 .GOTO 4822
4670 SEUIP  LDA $D011
4680 AND #$EF ;BILDSCHIRM AN
4690  STA $D011
4700  LDA #$0F
4710  JSR CLOSE
4720  LDA #$0F
4730  LDX #$08
4740  TAY
4750  JSR SETLFS
4760  LDA #$03
4770         LDX #<UIPLU
4780         LDY #>UIPLU
4790  JSR SETNAM
```

```
4800  JSR OPEN
4810  LDA #$0F
4820  JMP CLOSE
4822 :
4824 ;  WARTESCHLEIFE
4826 :
4830 WAIT  JSR GETBYT
4840  BEQ WAIT
4850  RTS
4852 :
4854 ;DISK INITIALISIEREN
4856 :
4860 INIT   LDA #$0F ;KOMMANDOKANAL
4870 LDX #$08 ;OEFFNEN
4880  TAY
4890  JSR SETLFS
4900 LDA #$01 ; "I"
4910        LDX #<STRI
4920        LDY #>STRI
4930  JSR SETNAM
4940  JMP OPEN
4942 :
4944 ;SPACE BZW. CARRIAGE RETURN SENDEN
4946 :
4950 SPACE  LDA #$20 ;SPACE
4960  .BYT $2C
4970 CARET  LDA #CR ;CARRIDGE RETURN
4980 JMP BASOUT ;SENDEN
4982 :
4984 ;AUSGABEROUTINE
4986 :
4990 AUSGAB JSR BASOUT
5000 STASTE TXA ;A,X,Y REGISTER RETTEN
5010  PHA
5020  TYA
5030  PHA
5040 JSR STOP ;<STOP>-TASTE ABFRAGEN
5050  CLC
5060        BNE WEITER
5070  SEC
5080 WEITER PLA ;REGISTER WIEDERHERSTELLEN
5090  TAY
5100  PLA
5110  TAX
5120 STOP2  BCS ENDE
5130 RETURN  RTS
5132 :
5134 ;       ABBRECHEN
5136 :
5140 ENDE  LDX #$F6
5150  TXS
5160 ;JSR SCRON
5170  LDA #$0F
```

```
5180  JSR CLOSE
5190  JSR CLALL
5200        JMP START
5202 :
5204 ;  EINGABEROUTINE
5206 :
5210 INPUT  JSR GETBYT
5220  BNE RETURN
5230  JSR STASTE
5240        BCC INPUT
5250 FEHLER LDX #$0F ;STATUS ABFRAGEN
5260  JSR SFINUM
5270  JSR SETPAR
5280  LDA GA
5290  JSR TALK
5300  LDA SA
5310  JSR SETALK
5320  JSR ASCHEX
5330  CMP #$20
5340  PHP
5350 BCC MELAUS ;UND BEI FEHLER
5360 PHA ;AUSGEBEN
5370  JSR CARET
5380  JSR CARET
5390  PLA
5400  JSR HEXASC
5410 MELAUS  JSR IECIN
5420  CMP #CR
5430  BEQ MELEND
5440  PLP
5450  PHP
5460  BCC MELAUS
5470  JSR BASOUT
5480  BCC MELAUS
5490 MELEND JSR UNTALK
5500 PLP : RTS
5502 TASTE  JSR PTASTE :JSR STOP2 : BNE TASTE
5504 LDA #7 :STA MOFLAG
5506 JMP MOTAUS
5511 : :
5512 ; AUSGABE STRINGS
5513 : :
5520 QUEINS  .BYT 13,13,13
5530 .ASC "QUELLDISK EINLEGEN !"
5540  .BYT 0
5550 ZICASS  .BYT 13,13
5560 .ASC "ZIELCASSETTE EINLEGEN!"
5570 .BYT 13,0
5580 TITEL .ASC "  BACKUP VON DISKETTE"
5590 .BYT 13,13
5600 .ASC   "      AUF CASSETTE"
5602 .BYT 13,13
5604 .ASC "MIT F A S T T A P E"
```

```
5606 .BYT 13,13,13
5608 .ASC " (C)    DIRK PAULISSEN"
5610 .BYT 13,13,13
5620 .ASC "      E = ENDE"
5630         .BYT 0
5640 SLONG   .BYT 13
5650         .ASC " ZU LANGE ZUM COPIEREN"
5660         .BYT 0
5670 LILONG  .BYT 13
5680         .ASC " LISTE IST VOLL"
5690         .BYT 0
5700 JANEIN  .ASC "JA/NEIN"
5710         .BYT 0
5720 STRJA   .ASC "  JA"
5730         .BYT 0
5740 STRILL  .BYT 13
5750         .ASC " ILLEGALER FILE-TYP"
5760         .BYT 0
5770 STRNO   .ASC " NEIN"
5780         .BYT 0
5790 FRAGE   .BYT 13,13,13
5800 .ASC "WOLLEN SIE DIE "
5802 .BYT 13
5804 .ASC "PROGRAMME"
5810 .BYT 13,13
5820 .ASC      "  1    DURCHGEHEND"
5830 .BYT 13,13
5840 .ASC      "  2    EINZELN"
5850 .BYT 13,13
5860 .ASC "COPIEREN ?"
5870         .BYT 0
5880 SREAD   .BYT 13,13
5890 .ASC "READING "
5900         .BYT 0
5910 PRGRE   .ASC ",P,R"
5920         .BYT 0
5930 UIMIN   .ASC "UI-"
5940         .BYT 0
5950 UIPLU   .ASC "UI+"
5960         .BYT 0
5970 STRI    .ASC "I"
5980         .BYT 0
5985 CATALO  .ASC "$"
5986 .BYT 0
5990 WRITI .BYT 13,13
6000 .ASC "SAVING "
6010 .BYT 0
6020 EOCOP   .BYT 13,13
6030 .ASC "ENDE DES COPIERENS"
6040         .BYT 0
6045 WMACH .BYT 13
6050         .ASC " WEITER MACHEN ? "
6060         .BYT 0
```

```
6070 SNFILE   .BYT 13,13
6080          .ASC "NAECHSTES FILE : "
6090 .BYT 13,0
6100 .END
```

Und hier der BASIC-Lader:

```
100 REM BACKUP DISKETTE => CASSETTE FUER C-64
110 E=51757:A=50176:PS=0
120 FORI=ATOE:READX:POKEI,X:PS=PS+X:NEXT
130 IFPS<>171785THENPRINT"FEHLER IN DATAS":END
140 SYSA:NEW
150 DATA169,1,141,32,208,141,33,208,169,6,141,134,2,169,107,160,200,32,3
0,171
160 DATA169,52,160,200,32,30,171,32,153,199,201,69,208,1,96,32,182,199,3
2,182
170 DATA199,32,159,199,169,8,170,160,0,32,186,255,169,1,162,225,160,201,
32
180 DATA189,255,32,192,255,169,8,32,180,255,169,0,32,150,255,160,4,32,16
5,255
190 DATA136,208,250,32,165,255,133,34,32,165,255,166,34,32,205,189,32,17
9,199
200 DATA32,165,255,240,6,32,210,255,24,144,245,32,182,199,32,182,199,32,
165
210 DATA255,32,165,255,160,0,132,251,32,165,255,133,252,32,165,255,133,2
53
220 DATA166,252,32,205,189,32,179,199,160,0,32,165,255,72,32,187,199,104
,153
230 DATA64,3,240,3,200,208,240,32,165,255,32,165,255,165,144,240,3,76,71
,197
240 DATA165,253,208,6,165,252,201,152,144,10,169,252,160,200,32,30,171,7
6,65
250 DATA197,165,251,201,48,144,10,169,27,160,201,32,30,171,24,144,237,16
9,0
260 DATA133,8,169,31,133,211,169,51,160,201,32,30,171,32,225,199,201,78,
240
270 DATA82,201,74,208,245,169,69,160,201,32,30,171,165,251,32,241,198,16
2,0
280 DATA232,189,64,3,201,34,208,248,134,253,232,189,64,3,201,34,240,6,14
5,65
290 DATA232,200,208,243,138,164,251,24,229,253,153,64,202,189,64,3,208,1
0,169
300 DATA81,160,201,32,30,171,76,65,197,201,83,240,242,201,80,240,3,232,2
08
310 DATA230,230,251,24,144,7,169,107,160,201,32,30,171,32,182,199,76,125
,196
320 DATA169,8,32,195,255,165,251,208,3,76,0,196,169,119,160,201,32,30,17
1,32
330 DATA225,199,201,49,240,7,201,50,208,245,169,0,44,169,255,133,254,169
,15
340 DATA32,195,255,162,0,134,252,169,199,160,201,32,30,171,164,252,190,6
4,202
```

```
350 DATA165,252,32,241,198,177,65,32,210,255,200,202,208,247,169,2,162,8
,168
360 DATA32,186,255,166,252,189,64,202,133,34,165,252,32,241,198,162,0,17
7,65
370 DATA157,64,3,200,232,198,34,208,245,160,0,185,210,201,157,64,3,200,2
32
380 DATA192,4,144,244,138,162,64,160,3,32,189,255,32,192,255,169,9,160,0
,132
390 DATA34,133,35,173,17,208,41,239,141,17,208,169,15,162,8,168,32,186,2
55
400 DATA169,3,162,215,160,201,32,189,255,32,192,255,162,2,32,15,243,32,3
1,243
410 DATA165,186,32,180,255,165,185,32,150,255,160,0,32,165,255,32,9,199,
166
420 DATA144,240,246,32,235,199,8,169,2,32,195,255,173,17,208,9,16,141,17
,208
430 DATA40,144,3,32,55,199,166,252,165,34,133,174,165,35,133,175,232,165
,252
440 DATA208,13,169,78,160,200,32,30,171,32,225,199,32,37,200,169,227,160
,201
450 DATA32,30,171,164,252,190,64,202,165,252,32,241,198,177,65,32,210,25
5,200
460 DATA202,208,247,162,1,169,0,168,32,186,255,166,252,189,64,202,133,34
,165
470 DATA252,32,241,198,162,0,177,65,157,64,3,232,200,198,34,208,245,138,
162
480 DATA64,160,3,32,189,255,160,0,169,9,132,172,133,173,177,172,133,167,
200
490 DATA177,172,133,168,200,132,172,165,174,56,229,172,8,24,101,167,133,
169
500 DATA165,175,101,168,40,229,173,133,170,162,5,134,171,32,133,192,32,2
29
510 DATA198,230,252,166,252,228,251,176,7,36,254,16,16,76,223,198,169,23
7,160
520 DATA201,32,30,171,32,225,199,76,0,196,32,85,199,169,78,160,200,32,30
,171
530 DATA32,225,199,32,37,200,32,182,199,76,117,197,32,115,199,173,17,208
,9
540 DATA16,141,17,208,96,160,0,10,10,132,66,10,38,66,10,38,66,133,65,165
,66
550 DATA24,105,203,133,66,160,0,96,145,34,230,34,208,2,230,35,88,96,32,1
65
560 DATA255,41,15,10,10,10,10,133,87,32,165,255,41,15,5,87,96,72,74,74,7
4,74
570 DATA32,47,199,104,41,15,24,105,48,76,210,255,173,17,208,9,16,141,17,
208
580 DATA169,7,160,202,32,30,171,32,225,199,201,89,208,1,96,201,78,208,24
4,76
590 DATA208,199,169,26,160,202,32,30,171,165,252,10,10,10,10,166,252,188
,64
600 DATA202,170,189,0,203,32,210,255,232,136,208,246,96,173,17,208,41,23
9,141
```

```
610 DATA17,208,169,15,32,195,255,169,15,162,8,168,32,186,255,169,3,162,2
19
620 DATA160,201,32,189,255,32,192,255,169,15,76,195,255,32,228,255,240,2
51
630 DATA96,169,15,162,8,168,32,186,255,169,1,162,223,160,201,32,189,255,
76
640 DATA192,255,169,32,44,169,13,76,210,255,32,210,255,138,72,152,72,32,
225
650 DATA255,24,208,1,56,104,168,104,170,176,1,96,162,246,154,32,229,198,
169
660 DATA15,32,195,255,32,231,255,76,0,196,32,228,255,208,233,32,190,199,
144
670 DATA246,162,15,32,15,243,32,31,243,165,186,32,180,255,165,185,32,150
,255
680 DATA32,19,199,201,32,8,144,11,72,32,182,199,32,182,199,104,32,38,199
,32
690 DATA165,255,201,13,240,9,40,8,144,245,32,210,255,144,240,32,171,255,
40
700 DATA96,32,56,248,32,205,199,208,248,169,7,133,192,76,202,252,13,13,1
3,18
710 DATA81,85,69,76,76,68,73,83,75,146,32,69,73,78,76,69,71,69,78,32,33,
0,13
720 DATA13,18,90,73,69,76,67,65,83,83,69,84,84,69,146,32,69,73,78,76,69,
71
730 DATA69,78,32,33,13,0,147,17,28,32,32,32,32,32,32,32,66,65,67,75,85,8
0,32
740 DATA86,79,78,32,68,73,83,75,69,84,84,69,32,65,85,70,13,13,31,32,32,3
2,32
750 DATA32,32,32,32,32,32,32,32,32,32,67,65,83,83,69,84,84,69,13,13,32,3
2,32
760 DATA32,32,32,32,32,77,73,84,32,18,129,70,32,65,32,83,32,84,32,84,32,
65
770 DATA32,80,32,69,32,146,13,13,13,32,32,32,40,67,41,32,32,32,32,68,73,
82
780 DATA75,32,80,65,85,76,73,83,83,69,78,32,32,13,13,13,30,32,32,32,32,3
2,32
790 DATA32,32,32,32,32,69,32,61,32,69,78,68,69,31,0,13,28,18,33,33,18,32
,90
800 DATA85,32,76,65,78,71,69,32,90,85,77,32,67,79,80,73,69,82,69,78,32,3
1,0
810 DATA13,28,18,33,33,18,32,76,73,83,84,69,32,73,83,84,32,86,79,76,76,3
2,31
820 DATA0,18,74,65,47,78,69,73,78,157,157,157,157,157,157,157,146,31,0,1
8,30
830 DATA32,32,74,65,32,32,146,32,31,0,13,28,18,33,33,18,32,73,76,76,69,7
1,65
840 DATA76,69,82,32,70,73,76,69,84,89,80,31,0,28,18,32,78,69,73,78,32,14
6,32
850 DATA31,0,13,13,13,87,79,76,76,69,78,32,83,73,69,32,68,73,69,32,80,82
,79
860 DATA71,82,65,77,77,69,13,13,32,32,32,32,49,32,32,32,32,68,85,82,67,7
2,71
```

```
870 DATA69,72,69,78,68,13,13,32,32,32,32,50,32,32,32,32,69,73,78,90,69,7
6,78
880 DATA13,13,67,79,80,73,69,82,69,78,32,63,0,13,13,82,69,65,68,73,78,71
,32
890 DATA0,44,80,44,82,0,85,73,45,0,85,73,43,0,73,0,36,0,13,13,83,65,86,7
3,78
900 DATA71,32,0,13,13,28,18,69,78,68,69,32,68,69,83,32,67,79,80,73,69,82
,69
910 DATA78,83,32,146,31,0,13,32,87,69,73,84,69,82,32,77,65,67,72,69,78,3
2,63
920 DATA32,0,13,13,78,65,69,67,72,83,84,69,83,32,70,73,76,69,32,58,32,0
```

10.14 Tips und Tricks zur Datasette

Die Datasette ist das wohl preiswerteste Speichergerät für den C64. Obwohl die Floppy bei den meisten Anwendern schon die Datasette vertrieben hat, möchte ich Ihnen dennoch ein paar Datasetten-Tips und -Tricks verraten, die Sie dann recht einfach auch in Ihre Programme einbauen können.

10.14.1 Steuern der Datasette von BASIC aus

Vielleicht ist Ihnen schon einmal die lästige Eigenschaft der Datasette aufgefallen, immer nach einem Lade- oder Speichervorgang selbsttätig zu stoppen. Um nun wieder ein Programm zu laden, muß zuerst die <Stop>-Taste und dann die <Play>-Taste gedrückt werden. Was stoppt denn eigentlich den Motor der Datasette? Nun, nichts anderes als der Computer. Das können wir nun ausnutzen und die Datasette vom Computer aus steuern!

Um dies zu demonstrieren, drücken Sie bitte einmal die <Play>-Taste und geben Sie im Direktmodus folgende zwei Befehle ein:

```
POKE 192,1
POKE 1,PEEK (1) OR 32
```

Und siehe da, die Datasette stoppt, als hätten Sie von Hand die <Stop>-Taste betätigt. Nun, wie funktioniert der Trick: Soll der Motor der Datasette eingeschaltet werden, so muß in der Speicherstelle 192 ein Wert ungleich Null stehen. Deshalb wird der Wert 1 in diese Speicherstelle geschrieben. Weiterhin muß das 5.

Bit der Speicherstelle 1 gesetzt werden, was mit dem Befehl POKE 1,PEEK (1) OR 32 geschieht. Ist in der Speicherstelle das Bit 5 gesetzt, so wird der Motor der Datasette abgeschaltet.

Es ist aber genauso möglich, den Motor starten zu lassen. Geben Sie dazu bitte folgende Befehle ein:

```
POKE 192,0
```

Es wird hier wieder in die Speicherstelle 192 der Wert 0 geschrieben. Mit den folgenden Befehlen läßt sich abfragen, ob eine Taste der Datasette gedrückt wurde oder nicht:

```
IF PEEK (1) = 55 THEN PRINT "KEINE TASTE GEDRÜCKT"
IF PEEK (1) =  7 THEN PRINT "TASTE GEDRÜCKT"
```

Wollen Sie einmal in einem Programm so lange warten, bis der Benutzer die <Stop>-Taste gedrückt hat, dann können Sie dies durch den Befehl

```
WAIT 1,16
```

erfahren. Dadurch wird der Computer nämlich veranlaßt, so lange zu warten, bis in Speicherzelle 1 das 4. Bit gesetzt ist, was nur dann der Fall ist, wenn die <Stop>-Taste der Datasette gedrückt wurde. Im folgenden habe ich Ihnen noch einmal alle Möglichkeiten zum Steuern der Datasette aufgelistet:

```
Motor aus:                POKE 192,1
                          POKE 1,PEEK (1) OR 32
Motor an:                 POKE 192,0
                          POKE 1,PEEK (1) AND 39
Warten auf <Stop>-Taste:  WAIT 1,16
Warten auf <Play>-Taste:  IF PEEK (1) = 55 THEN ...
```

10.14.2 Ein Kopierschutz für die Datasette

Wer hat nicht schon einmal ein Programm geschrieben, das er vor unerlaubtem Kopieren sichern wollte. Mit der Datasette ist es sehr einfach, einen Kopierschutz zu programmieren. Wird ein Programm auf Datasette abgesaved, so kann man einen Pro-

grammnamen von bis zu 172 Buchstaben Länge angeben. Beim Laden dieses Programms werden lediglich höchstens 16 Zeichen angegeben. "Doch was nützt uns das jetzt?" werden Sie sich sicher fragen. Nun, beim Laden eines Programms werden die restlichen 160 Buchstaben des Namens in den sogenannten Kassettenpuffer geladen. Kopiert sich nun jemand Ihr Programm, ohne daß Sie es wollen, so kennt er natürlich nicht den Rest des Namens. Beim Abspeichern gibt er deshalb nur einen höchstens 16 Zeichen langen Namen ein, denn woher soll er ahnen, daß das Programm einen 17 Zeichen langen Namen hat? Das können Sie sich zunutzen machen.

Ich habe hier nun für Sie ein kleines Programm geschrieben, welches das 17. Zeichen des Namens abfragt:

```
100 IF PEEK (849) = ASC("A") THEN SYS64738
110 PRINT "ES HANDELT SICH HIER UM EIN ORGINALPROGRAMM!"
120 PRINT "VIEL SPAß BEI DESSEN BENUTZUNG!"
```

Ab der Speicherstelle 849 befinden sich 160 Bytes, in denen der restliche Name des Programms steht. Dieser Teil des Namens läßt sich durch den PEEK-Befehl auslesen. Voraussetzung dafür ist jedoch, daß der Programmname 16 Zeichen zusätzlich der restlichen Zeichen lang ist. Ein Beispiel für einen solchen Namen ist

```
1234567890123456A
```

Die ersten 16 Zeichen werden beim Ladevorgang angezeigt, das siebzehnte Zeichen jedoch nicht. Es steht im Speicher an der Adresse 849. Durch die Befehlsreihe in Zeile 100 wird ein RESET ausgelöst, falls in der Speicherstelle 849 ein anderer Wert als der Code für das Zeichen A steht. Ansonsten fährt das Programm mit Zeile 110 fort. Sie sollten diese Zeilen immer vor den Programmanfang stellen. Sie können statt des Codes für das Zeichen A natürlich auch einen anderen Code abfragen. Dafür müssen Sie nur Zeile 100 abändern.

10.14.3 Beschleunigung des Ladevorgangs

Programme, die auf Datasette gespeichert werden, sind - um eventuellen Ladefehlern vorzubeugen - doppelt abgespeichert. Es ist jedoch Unsinn, ein Programm zweimal zu laden, da dies fast doppelt so lange dauert. Die Ladezeit läßt sich durch folgende Programmzeilen, die Sie an den Anfang des zu ladenden Programms stellen sollten, verkürzen.

```
POKE 45,PEEK (831)
POKE 46,PEEK (832)
CLR
```

Sie können nun nach dem ersten Ladevorgang die <Stop>-Taste drücken, damit das Programm nicht unnützerweise zweimal geladen wird. Durch die drei Zeilen wird bewirkt, daß die Zeiger auf das BASIC-Ende richtig gesetzt, und dann alle Variablen gelöscht werden. Würde dies nicht gemacht, so kann es leicht vorkommen, daß während des Programmablaufs einfach Fehler auftreten, die sonst nicht vorhanden sind.

Eine zweite Möglichkeit, den Ladevorgang zu beschleunigen, ist das Abbrechen der Wartepause, nachdem der Computer die Meldung ausgegeben hat, daß er das Programm gefunden hat. Dazu drücken Sie entweder die <Ctrl>-, die <Space>-, oder die <Commodore>-Taste. Ein Drücken des Feuerknopfes des Joysticks, der sich in Port 1 befindet, bewirkt das gleiche.

Manche Maschinenprogrammierer legen kleine Programmroutinen im Kassettenpuffer ab. Auch wenn man viel Speicherplatz für Sprite-Daten braucht, ist es sinnvoll, diese im Kassettenpuffer abzulegen. Viele Leute sind der Meinung, daß es dann nicht mehr möglich ist, Programme von der Datasette zu laden, da sonst die Maschinensprache-Routine überschrieben würde. Daß es dennoch möglich ist, von der Datasette zu laden, ohne Programme, die im Kassettenpuffer abgelegt sind, zu überschreiben, zeigen folgende zwei Befehle, die im Direktmodus eingegeben werden müssen:

```
POKE 178,0
POKE 179,4
```

Diese beiden POKEs bewirken, daß die Zeiger, die auf den Anfang des Kassettenpuffers zeigen, nun "umgebogen" werden und auf die Bildschirmspeicher zeigen. Dadurch werden die Daten, die sich im Kassettenpuffer befinden, nicht zerstört. Sie können natürlich auch andere Speicherbereiche als den Bildschirmspeicher angeben. Dazu müssen Sie nur die Parameter der POKE-Befehle abändern (siehe Anhang).

10.14.4 Retten eines Programms nach LOAD-ERROR

Die Datasette ist dafür bekannt, daß sie ein unsicheres Speichergerät ist. Es kommt manchmal vor, daß ein schlichter LOAD-ERROR nach einem versuchten Ladevorgang auftaucht. Doch noch müssen Sie nicht verzagen. Die Datasette speichert - wie schon erwähnt - jedes Programm doppelt ab. Beim Laden wird dann der erste Teil ganz normal geladen und mit dem zweiten Teil verglichen. Stimmen manche Teile nicht überein, so kommt es zu dem gefürchteten LOAD-ERROR. In Wirklichkeit ist der LOAD-ERROR also gar kein richtiger Ladefehler, sondern es sind lediglich die beiden Programmteile unterschiedlich.

Sollte einmal bei Ihnen ein LOAD-ERROR auftreten, so versuchen Sie das Programm zu listen. Wenn dies noch einigermaßen funktioniert, so ist Ihr Programm nicht verloren. Sie müssen nur noch den Zeiger auf das BASIC-Ende richtig setzen, da die Datasette diese erst dem Computer übergibt, wenn der Ladevorgang funktioniert hat, was bei einem LOAD-ERROR nicht der Fall ist. Die Werte für den Zeiger auf das BASIC-Ende finden Sie im Kassettenpuffer, und zwar bei Adresse 831/832. Diese Werte müssen lediglich dem Computer mitgeteilt werden, was durch folgende Befehle geschieht:

```
POKE 46,PEEK (832)
POKE 47,PEEK (831)
POKE 48,PEEK (832)
POKE 49,PEEK (831)
POKE 50,PEEK (832)
```

Nun läßt sich Ihr Programm wie gewöhnlich durch RUN starten und durch SAVE"PRG.Name",1 abspeichern. Da auch dieser

Trick nicht immer funktioniert, ist das sicherste Mittel gegen defekte Datenträger immer noch das Anfertigen einer Sicherheitskopie!

10.14.5 Sound aus der Datasette

Die Datasette arbeitet genauso wie ein ganz normaler Kassettenrekorder. Deshalb sollte es möglich sein, ihr auch einmal den neuesten Disco-Sound oder "Beethovens Neunte" zu entlocken. Daß dies möglich ist, zeigt - wie so oft - dieser POKE:

```
POKE 54296,15
```

Laden Sie nun ein Programm von Datasette, so werden Sie ein Gepiepse hören, was nichts anderes als die Daten ist, die momentan in den Rechner geladen werden. Der Trick funktioniert recht einfach: Genauso wie beim Fernseher, kann man auch beim SID (dies ist der Chip, der für den Sound zuständig ist) die Lautstärke regulieren. Dies geschieht im Register Nummer 25 des SID's. Dabei steht der Wert 15 für die höchste, und der Wert 0 für die niedrigste Lautstärke. Oben genannter POKE stellt also die Lautstärke auf "maximal" ein, wodurch die Töne der Datasette zu hören sind.

Es ist natürlich auch möglich, hier kleinere Werte einzugeben, was zur Folge hat, daß die Töne immer leiser werden. Sie sollten allerdings von der musikalischen Qualität der Datasette nicht viel erwarten, da sie nicht dafür gedacht ist, Musik wiederzugeben.

Anhang

Anhang A: BASIC-Referenz

Dieser Anhang enthält neben einer Liste der BASIC-Fehlermeldungen eine komplette Übersicht über die BASIC-2.0-Schlüsselworte. Wenn Ihnen die Bedeutung einer bestimmten BASIC-Anweisung nicht mehr ganz klar ist oder eine Fehlermeldung Ihnen Kopfzerbrechen bereitet, dann sollten Sie hier nachschlagen.

Alle Erklärungen sind jedoch - nicht zuletzt der besseren Übersichtlichkeit wegen - möglichst knapp gehalten.

Anhang A.1: Schlüsselworte

In der Kopfzeile steht jeweils der Name des Schlüsselwortes, seine Abkürzung (in geschweiften Klammern) sowie sein Typ (Befehl, Funktion, Operator oder reservierte Variable). Darunter folgt die Syntax. Optionale Parameter sind dabei in eckigen Klammern angegeben.

Die Syntax von Funktionen wird durch die Zuweisung an eine Variable verdeutlicht. In der Regel dürfen die Funktionen aber in beliebigen Ausdrücken verwendet werden. Sie sollten nur darauf achten, daß Sie eine numerische Funktion nicht in einem String-Ausdruck verwenden und umgekehrt.

Die Angabe "numerischer Ausdruck" in der Parameterliste besagt, daß an dieser Stelle ein beliebiger Ausdruck stehen darf, der einen numerischen Wert als Ergebnis liefert, beispielsweise 2*A/5 oder SIN(X+Y); die Angabe "String-Ausdruck" besagt, daß hier ein beliebiger Zeichenkettenausdruck stehen darf, beispielsweise "COMMODORE"+" 64" oder LEFT$(ZE$+WZ$,7).

ABS {A<Shift>+B}	Funktion

`A = ABS (numerischer Ausdruck)`

ABS berechnet den Absolutwert (Betrag) eines in Klammern angegebenen numerischen Ausdrucks. Der Betrag einer Zahl ist immer positiv. ABS empfiehlt sich daher immer dann, wenn man sicher gehen möchte, daß ein Ausdruck nicht negativ wird (etwa bei POKE).

AND {A<Shift>+N}	Operator

`A = logischer/numerischer Ausdruck AND log./num. Ausdruck`

AND verknüpft zwei Ausdrücke bitweise durch ein logisches UND. Jeder Ausdruck wird dazu intern in eine 16-Bit-Zahl umgerechnet. AND eignet sich vor allem zum gezielten Löschen einzelner Bits, etwa der eines Registers.

ASC {A<Shift>+S}	Funktion

`A = ASC (String-Ausdruck)`

ASC wandelt das erste Zeichen des angegebenen String-Ausdrucks in seinen ASCII-Wert um (siehe auch Anhang C.10). Die Umkehrfunktion zu ASC ist → CHR$.

ATN {A<Shift>+T}	Funktion

`A = ATN (numerischer Ausdruck)`

ATN berechnet den Arcustangens des angegebenen numerischen Ausdrucks. Der Arcustangens ist derjenige Winkel, dessen Tangenswert gleich dem Wert des numerischen Ausdrucks ist. ATN

wird im Normalfall kaum benötigt, kann aber bei speziellen mathematischen Berechnungen sehr nützlich sein.

CHR$ {C<Shift>+H} — Funktion

```
A$ = CHR$(numerischer Ausdruck)
```

CHR$ weist A$ das dem Wert des numerischen Ausdrucks (0 bis 255) entsprechende ASCII-Zeichen zu (siehe auch Anhang C.10). CHR$ ist vor allem bei der Arbeit mit speziellen Tasten (z.B. <Return> oder den Funktionstasten) oder mit Grafikzeichen sehr hilfreich. Die Umkehrfunktion zu CHR$ ist → ASC.

CLOSE {CL<Shift>+O} — Befehl

```
CLOSE logische File-Nummer
```

CLOSE schließt eine zuvor mittels → OPEN geöffnete logische Datei. Dazu muß die logische File-Nummer der Datei (0 bis 255) angegeben werden. CLOSE sollte den Abschluß jedes Dateizugriffs bilden. Das Vergessen der CLOSE-Anweisung kann zu einem Datenverlust führen!

CLR {C<Shift>+L} — Befehl

```
CLR
```

CLR löscht sämtliche momentan im BASIC-Speicher vorhandenen Variablen und setzt diverse Zeiger zurück. Das momentan im Rechner befindliche Programm bleibt aber unverändert. Um auch dieses zu löschen, müssen Sie → NEW eingeben.

CMD {C<Shift>+M} **Befehl**

```
CMD logische File-Nummer [, beliebiger Ausdruck]
```

CMD leitet die Bildschirmausgabe auf die durch ihre logische File-Nummer bezeichnete Datei um. Hinter der File-Nummer kann ein beliebiger Ausdruck stehen, der in die Datei geschrieben werden soll. Alle nachfolgenden PRINT- oder LIST-Anweisungen werden ebenfalls in die Datei geschrieben. Ein "PRINT#log. File-Nr." beendet die Umlenkung. Die Ausgabe erfolgt nun wieder auf den Bildschirm. (Bitte vergessen Sie nicht, die Datei mit CLOSE zu schließen). CMD wird in den meisten Fällen im Zusammenhang mit dem Drucker eingesetzt.

"OPEN 4,4:CMD 4:LIST:PRINT#4:CLOSE 4" zum Beispiel gibt das momentan im Speicher befindliche BASIC-Programm auf dem Drucker aus.

CONT {C<Shift>+O} **Befehl**

```
CONT
```

CONT setzt die Ausführung eines BASIC-Programms, das durch die <Stop>-Taste oder eine STOP-Anweisung im Programm unterbrochen wurde, wieder fort. Voraussetzung dafür ist allerdings, daß Sie am Programm in der Zwischenzeit keine Veränderungen vorgenommen haben. Sonst bekommen Sie einen CAN'T CONTINUE ERROR.

DATA {D<Shift>+A} **Befehl**

```
DATA Wert1,Wert2,.....
```

DATA dient zur Speicherung numerischer Daten oder String-Daten innerhalb eines Programms. Diese können dann bei Bedarf mit → READ gelesen werden. Die einzelnen Daten müssen durch

Kommas voneinander getrennt werden. Die Anzahl der DATA-Zeilen ist nur durch den verfügbaren BASIC-Speicher begrenzt. Die DATA-Zeilen dürfen sich außerdem an beliebiger Stelle innerhalb eines Programms befinden.

DEF {D<Shift>+E}	**Befehl**

```
DEF FN Name(X) = numerischer Ausdruck
```

DEF dient zur Definition einer numerischen Funktion, die dann durch FN aufgerufen werden kann. Für den Funktionsnamen gilt dasselbe wie für Variablennamen, d.h., er darf maximal zweistellig sein und muß mit einem Buchstaben beginnen. Zu beachten ist, daß die DEF-Definition vor dem ersten FN-Aufruf abgearbeitet sein muß. Die Definition sollte daher am Anfang eines Programms stehen oder in einem Unterprogramm, das dann zu Beginn aufgerufen wird.

DEF bietet sich vor allem bei sehr häufig benötigten Berechnungen an. Die Berechnungsformel wird dazu einmal definiert (z. B. DEF FN MW(X) = X*0.14) und dann jeweils mit dem zu berechnenden Wert aufgerufen (z. B. "PRINT FN MW(100)" ergibt "14"). Das erhöht nicht nur die Übersichtlichkeit im Programm, sondern läßt sich auch wesentlich leichter warten.

DIM {D<Shift>+I}	**Befehl**

```
DIM Feldname1, Feldname2,.....
```

Größere Variablenfelder müssen vor ihrer ersten Verwendung mit DIM vordimensioniert werden. Dies gilt für alle Felder, die mehr als 11 Elemente enthalten sollen. Die Dimensionierung erfolgt durch Angabe des höchsten Feldelements. "DIM A(100)" beispielsweise definiert ein 101 Elemente umfassendes Feld, wobei der Index von A(0) bis A(100) läuft. Hinter einer DIM-An-

weisung dürfen so viele Dimensionierungen stehen, wie in eine Programmzeile passen, jeweils getrennt durch ein Komma.

Taucht eine Feldvariable in einem Programm auf, ohne vorher dimensioniert worden zu sein, so wird die Feldgröße auf 11 Elemente festgelegt. Eine nachträgliche Änderung der Dimensionierung ist nicht möglich! Daher muß man sich von vornherein genau überlegen, wie groß das Feld eventuell werden könnte.

END {E<Shift>+N}	Befehl

```
END
```

END bildet den logischen Abschluß eines BASIC-Programms. END ist im Normalfall nicht unbedingt erforderlich (Der BASIC-Interpreter verzweigt nach Abarbeitung der letzten Programmzeile auch von selbst in den Direktmodus.), sollte aber der Ordnung halber nicht weggelassen werden. Insbesondere dann, wenn Sie in Ihrem Programm Unterprogramme verwenden, die ganz am Ende stehen. Ein RETURN WITHOUT GOSUB ERROR ist die häufige Folge eines vergessenen END, wenn der Interpreter am Programmende im ersten Unterprogramm auf ein RETURN trifft, ohne daß zuvor ein GOSUB-Aufruf erfolgte. Nach END stehen das Programm und sämtliche Variablen weiterhin zur Verfügung.

EXP {E<Shift>+X}	Funktion

```
A = EXP(Potenz)
```

EXP berechnet die Exponentialfunktion mit der Eulerschen Zahl e (=2.718...) als Basis. Die Potenz darf Werte von etwa -88 (e^-88) bis +88 (e^+88) annehmen. Die Umkehrfunktion zu EXP ist → LOG.

FOR {F<Shift>+O}	Befehl

```
FOR SC = Startwert TO Endwert [STEP Schrittweite]
..... (beliebige Anweisungen)
NEXT SC
```

Mit FOR lassen sich Programmschleifen realisieren. Die zwischen FOR und NEXT stehenden Anweisungen werden so oft wiederholt, bis der Wert der Variablen SC den hinter TO stehenden Endwert übersteigt. Normalerweise wird SC durch NEXT jeweils um eins erhöht. Durch STEP kann jedoch eine beliebige, auch negative Schrittweite definiert werden.

Es ist auch möglich, FOR ... NEXT-Anweisungen zu schachteln, allerdings nur bis zu einer Verschachtelungstiefe von etwa drei bis fünf, je nachdem, ob innerhalb der Schleifen auch noch Unterprogrammaufrufe enthalten sind. Ein OUT OF MEMORY ERROR deutet in diesem Zusammenhang darauf hin, daß die Schleifen zu tief verschachtelt wurden.

FRE {F<Shift>+R}	Funktion

```
A = FRE(0)
```

FRE berechnet den momentan noch freien (d.h. von Programm und Variablen nicht genutzten) BASIC-Speicherplatz. Maximal stehen etwa 39.000 Bytes für BASIC zur Verfügung. Die Zahl in Klammern hinter FRE ist ohne Bedeutung, muß aber angegeben werden. In der Regel verwendet man eine Null.

FRE hat die unangenehme Eigenschaft, daß es bei einem freien Speicher von mehr als 32.767 Bytes einen negativen Wert liefert zu dem man 65.536 addieren muß, um den korrekten Wert zu erhalten. Hier kann man sich aber durch einen kleinen Trick behelfen: Mit "FRE(0)-(FRE(0)<0)*65536" erhält man immer die wahre Speichergröße.

GET {G<Shift>+E} Befehl

```
GET EG/EG$ [,weitere Variablen]
```

GET liest ein einzelnes Zeichen von der Tastatur ein und weist es der Variablen EG bzw EG$ zu. Im Gegensatz zu INPUT wartet GET nicht auf eine Eingabe. Erfolgt zum Zeitpunkt der Abfrage keine Eingabe, so wird EG der Wert Null bzw. EG$ eine leere Zeichenkette zugewiesen.

GET# {G<Shift>+E#} Befehl

```
GET# logische File-Nummer,EG/EG$ [,weitere Variablen]
```

GET# liest ein einzelnes Zeichen aus einer durch ihre logische File-Nummer bezeichneten Datei und weist es EG bzw. EG$ zu. Im Gegensatz zu INPUT# lassen sich mit GET# jede Art von Zeichen einlesen, also auch RETURN, Doppelpunkte oder Anführungszeichen.

GOSUB {GO<Shift>+S} Befehl

```
GOSUB Zeilennummer
```

Nach einem GOSUB unterbricht der BASIC-Interpreter die laufende Programmabarbeitung, springt zu der angegebenen Programmzeile und arbeitet das dort stehende Unterprogramm ab. Nach RETURN, am Ende des Unterprogramms, fährt der Interpreter hinter der GOSUB-Anweisung mit der Programmabarbeitung fort. Unterprogrammaufrufe dürfen auch verschachtelt werden, d.h., aus einem Unterprogramm heraus kann ein weiteres Unterprogramm aufgerufen werden. Bei einer zu hohen Verschachtelungstiefe (maximal etwa fünf) erhält man aber einen OUT OF MEMORY ERROR.

GOTO {G<Shift>+O} **Befehl**

```
GOTO Zeilennummer
```

GOTO springt zu der durch ihre Nummer bezeichneten BASIC-Zeile. Die Zeilennummer muß dazu konkret in Ziffern angegeben werden. Die Verwendung von Variablen oder numerischen Ausdrücken ist nicht erlaubt.

GOTO ist eine Anweisung, die man nur äußerst sparsam verwenden sollte, da sie sehr schnell zu sog. "Spaghetti-Code" führt.

Damit meint man an sich unbegründete Sprünge in einem BASIC-Programm, die, würde man Sprungstelle und Sprungziel durch eine Linie verbinden, an einen Teller "Spaghetti" erinnern.

IF {-} **Befehl**

```
IF Bedingung THEN Zeilennummer
IF Bedingung GOTO Zeilennummer
IF Bedingung THEN ..... (beliebige Anweisungen)
```

IF erlaubt die bedingte Verzweigung zu einer durch ihre Nummer bezeichneten BASIC-Zeile bzw. die bedingte Abarbeitung der hinter THEN stehenden Anweisungen.

Ist die zwischen IF und THEN bzw. GOTO stehende Bedingung logisch "WAHR" (z. B. "A<10", wobei A=2), so wird die angegebene Progammzeile angesprungen bzw. die hinter THEN stehenden Anweisungen ausgeführt.

Ist die Bedingung logisch "FALSCH" (z.B. "A>10", wobei A=2), so ignoriert der Interpreter den Rest der Programmzeile und fährt in der nächsten mit der Programmabarbeitung fort.

INPUT {-}	Befehl

`INPUT [Hinweistext,] Eingabevariable, .....`

INPUT dient zur Eingabe beliebiger numerischer oder alphanumerischer Daten. Die Eingabe wird an eine oder mehrere hinter INPUT stehende Variablen zugewiesen. Die Daten werden jeweils zunächst am Bildschirm ausgegeben, so daß sich Fehleingaben leicht korrigieren lassen. Erst durch Drücken der <Return>-Taste werden die Daten in die Variable übernommen und das Programm fortgesetzt. Wahlweise kann hinter INPUT ein Hinweistext angegeben werden, der dem Anwender sagt, welche Daten von ihm gefordert werden (beispielsweise "Bitte Namen eingeben:").

INPUT# {I<Shift>+N}	Befehl

`INPUT# logische File-Nr., Eingabevariable, .....`

INPUT# liest aus einer durch ihre logische File-Nummer bezeichneten Datei beliebige Daten, die durch RETURNs (CHR$(13); wird von PRINT# automatisch erzeugt), Kommas oder Doppelpunkte voneinander getrennt sein müssen, in die angegebenen Variablen ein. INPUT# ist allerdings nur in der Lage, String-Daten mit einer Länge bis zu 88 Zeichen einzulesen. Längere Zeichenketten verursachen einen STRING TOO LONG ERROR. In diesem Fall muß man sich mit GET# behelfen und die Daten byteweise einlesen.

INT {-}	Funktion

`A = INT (numerischer Ausdruck)`

INT berechnet den Integer-Wert, d.h. den ganzzahligen Anteil, des angegebenen numerischen Ausdrucks. INT(2.8) zum Beispiel ergibt 2, INT(-4.1) dagegen ist -5. INT rundet also immer auf

die nächst kleinere Ganzzahl ab. Um eine "echte" (kaufmännische) Rundung zu erreichen, addiert man zu der zu rundenden Zahl einfach 0.5. INT(A+0.5) beispielsweise rundet die in A gespeicherte Zahl immer korrekt.

LEFT$ {LE<Shift>+F}	Funktion

```
A$ = LEFT$ (String-Ausdruck,Länge)
```

LEFT$ ergibt einen Teil-String, der einen linken Teil des angegebenen String-Ausdrucks enthält, wobei Länge die Anzahl der "abzuschneidenden" Zeichen angibt. "A$ = LEFT$ ("BEISPIEL",3)" zum Beispiel weist A$ den Teil-String "BEI" zu. Der String-Ausdruck selbst bleibt durch die Operation unverändert.

LEN {-}	Funktion

```
A = LEN (String-Ausdruck)
```

LEN ermittelt die Länge des angegebenen String-Ausdrucks. Die Länge eines Strings liegt immer im Bereich zwischen 0 und 255.

LET {L<Shift>+E}	Befehl

```
LET A = numerischer Ausdruck
LET A$ = String-Ausdruck
```

LET weist einer Variablen den hinter dem Gleichheitszeichen angegebenen Ausdruck zu. LET ist ein Befehl, den man an sich überhaupt nicht benötigt, da Wertezuweisungen an Variablen vom BASIC-Interpreter auch ohne vorangestelltes LET angenommen werden.

LIST {L<Shift>+I}	Befehl

`LIST [Zeilennummer-] [-Zeilennummer]`

LIST gibt das ganze oder einen Teil des momentan im Speicher befindlichen BASIC-Programms auf dem aktuellen Ausgabegerät (in der Regel der Bildschirm; durch CMD kann die Ausgabe jedoch umgeleitet werden) aus. Sollen nur Teile eines Programms gelistet werden, so müssen die Nummern der gewünschten Zeilen hinter LIST angegeben werden. LIST 100-200 beispielsweise gibt die Programmzeilen 100 bis 200 aus, LIST -100 listet alle Programmzeilen vom Programmstart bis einschließlich Zeile 100, LIST 200- gibt alle Programmzeilen ab Zeile 200 bis zum Programmende aus.

LOAD {L<Shift>+O}	Befehl

`LOAD Name,Geräteadresse,Sekundäradresse`

LOAD lädt die mit ihrem Namen bezeichnete Datei (in den meisten Fällen ein BASIC-Programm) von Kassette oder Diskette. Um von Kassette zu laden, geben Sie als Geräteadresse eine 1 an. Die Geräteadresse der Floppy ist in der Regel 8. Falls Sie mit mehreren Floppies arbeiten, kann es aber auch ein Wert zwischen 9 und 15 sein. Die Sekundäradresse ist entweder 0 (falls Sie ein BASIC-Programm laden wollen) oder 1 (falls die Datei andere Daten enthält, etwa eine Grafik oder ein Maschinenprogramm).

MID$ {M<Shift>+I}	Funktion

`A$ = MID$ (String-Ausdruck, Startstelle, Länge)`

MID$ "schneidet" aus dem angegebenen String-Ausdruck einen Teil-String aus, wobei "Startstelle" die Position des ersten Zei-

chens und "Länge" die Anzahl der zu entnehmenden Zeichen angibt. "MID$ ("BEISPIEL",4,3)" zum Beispiel ergibt "SPI". Der String-Ausdruck selbst bleibt durch die Operation unverändert.

NEXT {N<Shift>+E}	Befehl

→ FOR.

NOT {N<Shift>+O}	Operator

```
A = NOT logischer/numerischer Ausdruck
```

NOT ermittelt das logische NICHT eines Ausdrucks. Logisch "FALSCHE" Ausdrücke werden daher "WAHR" und umgekehrt. Numerische Ausdrücke werden zunächst in eine 16-Bit-Zahl umgerechnet und dann jedes einzelne Bit "umgedreht" (falls es 0 war, wird es 1 und umgekehrt).

ON {-}	Befehl

```
ON Verteilerwert GOTO Zeilennummer,.....
ON Verteilerwert GOSUB Zeilennummer,.....
```

ON erlaubt - ähnlich wie IF - die bedingte Verzweigung in eine der hinter GOTO bzw. GOSUB angegebenen Zeilennummern. Im Gegensatz zu IF hat man aber die Möglichkeit mehrere Sprungziele in einer Anweisung zusammenzufassen. Verzweigt wird jeweils in Abhängigkeit vom Verteilerwert. Eine 1 hat zur Folge, daß in die erste Zeilennummer hinter GOTO bzw. GOSUB gesprungen wird, bei einer 2 wird in die zweite Zeilennummer verzweigt usw... Bei ungültigen Werten wird das Programm mit der hinter ON folgenden Anweisung fortgesetzt.

OPEN {O<Shift>+P}	Befehl

```
OPEN logische File-Nr.,Geräteadr.,Sekundäradr.,Kommandotext
```

OPEN öffnet eine logische Datei zum Lesen und/oder Schreiben von Daten von einem Peripheriegerät. Dazu muß die Adresse des Geräts (Floppy z. B. 8) sowie im Kommandotext der Name der Datei angegeben werden.

Der Kommandotext kann zusätzlich noch andere Parameter enthalten, etwa den Programmtyp bei einer Floppy-Datei. Die logische File-Nummer und die Sekundäradresse (beide dürfen Werte zwischen 0 und 255 annehmen) dienen der Unterscheidung der einzelnen geöffneten Dateien (bis zu zehn Dateien dürfen gleichzeitig geöffnet sein).

Bei allen späteren Zugriffen auf die Datei (durch INPUT#, PRINT# usw.) muß jeweils die logische File-Nummer zur Identifikation angegeben werden. Insbesondere bei Floppy-Dateien ist es sehr wichtig, daß die einzelnen Dateien auch verschiedene Sekundäradressen haben, da die Floppy diese zur internen Verwaltung der Datenkanäle benötigt.

Am einfachsten ist es, wenn man für die logische File-Nummer und die Sekundäradresse jeweils die gleichen Werte nimmt. "OPEN 2,8,2,"TEST,S,W"" z.B. öffnet eine sequentielle Datei namens TEST zum Schreiben von Daten.

OR {-}	Operator

```
A = logischer/numerischer Ausdruck OR log./num. Ausdruck
```

OR verknüpft zwei Ausdrücke bitweise durch logisches ODER. Jeder Ausdruck wird dazu intern in eine 16-Bit-Zahl umgerechnet. AND eignet sich vor allem zum gezielten Setzen einzelner Bits, etwa der eines Registers.

PEEK {P<Shift>+E} Funktion

```
A = PEEK(Speicheradresse)
```

PEEK dient zum Auslesen des Inhalts von Speicherstellen. Dazu muß die Adresse der Speicherstelle angegeben werden. Werte von 0 bis 65535 sind erlaubt. PEEK bildet das Gegenstück zu POKE.

POKE {P<Shift>+O} Befehl

```
POKE Speicheradresse, Wert
```

POKE schreibt den angegebenen Wert (0 bis 255) in die durch ihre Adresse (0 bis 65535) bezeichnete Speicherstelle. Das Gegenstück zu POKE bildet die Funktion PEEK. POKE und PEEK (in Verbindung mit SYS) eignen sich sehr gut zur Kommunikation mit Maschinenprogrammen.

POS {-} Funktion

```
A = POS (0)
```

POS übergibt die aktuelle Spaltenposition des Cursors am Bildschirm und kann daher bei der Programmierung von Bildschirmmasken recht hilfreich sein.

PRINT {?} Befehl

```
PRINT beliebiger Ausdruck
```

PRINT dient zur Ausgabe von Zahlen und Zeichenketten jeglicher Art und ist neben INPUT der wohl am häufigsten benötigte Befehl. Mit Hilfe spezieller Steuerzeichen, etwa CHR$(147) zum

Löschen des Bildschirms (siehe auch die ASCII-Tabelle im Anhang), läßt sich über PRINT auch das Erscheinungsbild der Bildschirmausgaben verändern. Die auszugebenden Ausdrücke müssen jeweils durch ein Komma oder ein Semikolon voneinander getrennt werden. Zum Beispiel löscht PRINT CHR$(147);"HALLO" zunächst den Bildschirm und schreibt dann den Text "HALLO" in die oberste Bildschirmzeile.

PRINT# {P<Shift>+R}	**Befehl**

```
PRINT# logische File-Nummer,beliebiger Ausdruck
```

PRINT# hat dieselbe Funktion wie PRINT, nur daß die Ausgabe nicht auf den Bildschirm, sondern in eine durch ihre logische File-Nummer bezeichnete Datei erfolgt. Der Aufbau des auszugebenden Ausdrucks ist analog zu PRINT, allerdings haben die Steuerzeichen bei PRINT# keine unmittelbare Wirkung (mit Ausnahme von RETURN (CHR$(13)), das zur Trennung zweier Datensätze dient), werden aber trotzdem (in Form ihrer ASCII-Werte) geschrieben.

READ {R<Shift>+E}	**Befehl**

```
READ Einlesevariable,.....
```

READ liest in DATA-Anweisungen abgelegte, numerische oder alphanumerische Daten in eine oder mehrere Variablen ein. Nach jeder READ-Anweisung wird der Zeiger auf die Daten automatisch um ein Element weitergesetzt. Nach Erreichen des letzten im Programm vorhandenen DATA-Elements erscheint ein OUT OF DATA ERROR. Mittels RESTORE läßt sich der DATA-Zeiger auf das erste im Programm vorhandene DATA-Element zurücksetzen.

REM {-} **Befehl**

```
REM Kommentar
```

REM dient zur Markierung von Kommentaren in einem Programm. Hinter REM dürfen beliebige Texte stehen. Der Rest der Programmzeile hinter REM wird vom BASIC-Interpreter überlesen. BASIC-Befehle, die hinter REM stehen (auch solche, die durch einen Doppelpunkt abgetrennt sind) werden daher grundsätzlich nicht abgearbeitet.

RESTORE {RE<Shift>+S} **Befehl**

RESTORE setzt den DATA-Zeiger auf das erste im Programm vorkommende DATA-Element. Eine nachfolgende READ-Anweisung liest dann dieses Element in eine Variable ein.

RESTORE ist insbesondere dann erforderlich, wenn in DATA-Anweisungen abgelegte Daten während des Programmablaufs mehrfach gelesen werden sollen.

RETURN {RE<Shift>+T} **Befehl**

RETURN bildet den logischen Abschluß eines durch GOSUB aufgerufenen Unterprogramms.

Nach RETURN fährt der BASIC-Interpreter hinter der zuletzt abgearbeiteten GOSUB-Anweisung mit der Programmabarbeitung fort.

RIGHT$ {R<Shift>+I}	Funktion

```
A$ = RIGHT$ (String-Ausdruck,Länge)
```

RIGHT$ ergibt einen Teil-String, der einen rechten Teil des angegebenen String-Ausdrucks enthält, wobei "Länge" die Anzahl der "abzuschneidenden" Zeichen angibt. "A$ = RIGHT$ ("BEISPIEL",5)" zum Beispiel weist A$ den Teil-String "SPIEL" zu. Der String-Ausdruck bleibt durch die Operation unverändert.

RND {R<Shift>+N}	Funktion

```
A = RND (numerischer Ausdruck)
```

RND erzeugt eine Zufallszahl im Bereich zwischen 0 und 1. Mit Hilfe einer einfachen Formel lassen sich daraus Zufallszahlen innerhalb eines beliebigen Bereichs erzeugen: UG+RND(1)*(OG-UG). UG steht für die untere, OG für die obere Grenze des Bereichs. 1+RND(1)*9 beispielsweise liefert Zufallszahlen im Bereich zwischen 1 und 10.

Das Vorzeichen des numerischen Ausdrucks hinter RND legt fest, welcher Startwert zur Berechnung der Zufallszahlen genommen wird. Ein positiver Wert hat zur Folge, daß ein intern im ROM des Rechners gespeicherter Wert als Basis genommen wird.

Der Wert "Null" sorgt dafür, daß der Basiswert aus einem CIA-Register geholt wird, wodurch sich wirklich "zufällige" Zahlen erzeugen lassen.

Bei einem negativen Wert schließlich wird in Abhängigkeit vom Wert des numerischen Ausdrucks selbst ein Basiswert generiert.

RUN {R<Shift>+U}	Befehl

```
RUN [Zeilennummer]
```

RUN startet ein im Speicher befindliches BASIC-Programm. Ohne Angabe einer Zeilennummer hinter RUN wird das Programm von Anfang an abgearbeitet; andernfalls wird ab der angegebenen Zeile begonnen. Zuvor werden in jedem Fall sämtliche evtl. aus früheren Programmläufen noch vorhandenen Variableninhalte gelöscht.

SAVE {S<Shift>+A}	Befehl

```
SAVE Name, Geräteadresse [,Sekundäradresse]
```

SAVE speichert das momentan im Speicher befindliche BASIC-Programm auf Kassette (Geräteadresse 1) oder Diskette (Geräteadresse 8). Die Sekundäradresse hat nur beim Speichern auf Kassette eine Bedeutung. Durch Angabe einer 2 oder 3 erreichen Sie, daß am Ende des Speichervorgangs zusätzlich eine Bandende-Markierung geschrieben wird.

SGN {S<Shift>+G}	Funktion

```
A = SGN (numerischer Ausdruck)
```

SGN bestimmt das Vorzeichen des angegebenen numerischen Ausdrucks. Bei positiven Werten wird dabei eine 1, bei negativen Werten eine -1 und bei einer Null eine 0 übergeben.

SIN {S<Shift>+I} **Funktion**

```
A = SIN (numerischer Ausdruck)
```

SIN berechnet den Sinus eines im Bogenmaß angegebenen numerischen Ausdrucks. Die auch übliche Angabe in Grad läßt sich durch Multiplikation mit PI/180 ins Bogenmaß umrechnen.

SPC {S<Shift>+P} **Funktion**

```
PRINT SPC(Anzahl)
```

SPC in Verbindung mit PRINT verschiebt den Cursor um die angegebene Anzahl Spaltenpositionen (0 bis 255) nach rechts. SPC kann beim Aufbau von Bildschirmmasken sehr hilfreich sein.

SQR {S<Shift>+Q} **Funktion**

```
A = SQR (numerischer Ausdruck)
```

SQR berechnet die sog. Quadratwurzel des angegebenen numerischen Ausdrucks, der größer oder gleich Null sein muß.

ST {-} **Reservierte Variable**

ST ist eine reservierte Variable, die nach jeder Ein-/Ausgabeoperation den aktuellen Status anzeigt, etwa "Dateiende erreicht" oder "Schreibfehler".

STEP {ST<Shift>+E}	Befehl

→ FOR

STOP {S<Shift>+T}	Befehl

```
STOP
```

STOP unterbricht - ähnlich der <Stop>-Taste - das laufende Programm und kehrt in den Direktmodus zurück. Sofern Sie das Programm nicht verändern, können Sie es anschließend durch Eingabe von CONT hinter der Unterbrechungsstelle fortsetzen lassen. STOP kann beim Austesten von Programmen sehr hilfreich sein, etwa, indem man sich nach STOP bestimmte Variableninhalte ausgeben läßt.

STR$ {ST<Shift>+R}	Funktion

```
A$ = STR$ (numerischer Ausdruck)
```

STR$ wandelt eine Zahl in eine Zeichenkette um, beispielsweise um diese anschließend besser formatieren zu können. STR$ bildet das Gegenstück zu der Funktion VAL.

SYS {S<Shift>+Y}	Befehl

```
SYS Speicheradresse
```

SYS ruft das ab der angegebenen Speicheradresse (theoretisch 0 bis 65.535) beginnende Maschinenprogramm auf. Nach Abarbeitung des Maschinenprogramms wird das BASIC-Programm hinter der SYS-Anweisung fortgesetzt. Ergänzend zu SYS gibt es die Funktion USR, mit der sich Maschinenprogramme in BASIC als Funktion eingliedern lassen.

TAB {T<Shift>+A} Funktion

```
PRINT TAB(Position)
```

TAB in Verbindung mit PRINT setzt den Cursor auf die angegebene Spaltenposition (0 bis 255). Dazu wird der Cursor von seiner momentanen Position aus um eine entsprechende Anzahl Schritte nach rechts verschoben. Als Ergänzung zu TAB gibt es die Funktion SPC. Beide können beim Aufbau von Bildschirmmasken oder Tabellen sehr hilfreich sein.

TAN {-} Funktion

```
A = TAN (numerischer Ausdruck)
```

TAN berechnet den Tangens eines im Bogenmaß angegebenen numerischen Ausdrucks. Die auch übliche Angabe in Grad läßt sich durch Multiplikation mit PI/180 ins Bogenmaß umrechnen.

THEN {T<Shift>+H} Befehl

→ IF

TI/TI$ {-} Reservierte Variable

TI und TI$ sind zwei reservierte Variablen, die zum Stellen bzw. Lesen der internen Uhr des C64 dienen. Diese Uhr ist allerdings rein software-mäßig realisiert (sie wird über den Interrupt weitergestellt) und daher in der Regel sehr ungenau. Zum Stellen der Uhr muß TI$ ein sechs Zeichen langer String übergeben werden: TI$="HHMMSS". "HH" steht für die Stunden, "MM" für die Minuten, "SS" für die Sekunden. "153020" beispielsweise stellt die Uhr auf 15 Uhr 30 Minuten und 20 Sekunden. Zum Lesen

der Uhr können Sie TI$ oder TI verwenden. TI$ enthält die aktuelle Uhrzeit in demselben Format, wie beim Stellen der Uhr, also "HHMMSS". TI dagegen enthält die Uhrzeit in 1/60 Sekunden und eignet sich daher zum Messen sehr kurzer Zeitabstände.

USR {U<Shift>+S} — Funktion

```
A = USR (numerischer Ausdruck)
```

USR erlaubt den Aufruf eines Maschinenprogramms als BASIC-Funktion. Die Adresse des Maschinenprogramms muß dazu in den Speicherzellen 785 und 786 abgelegt werden. Anschließend kann USR wie jede andere BASIC-Funktion verwendet werden. Der Wert des hinter USR angegebenen numerischen Ausdrucks wird jeweils an das Maschinenprogramm übergeben, von diesem verarbeitet und schließlich ein Ergebniswert an BASIC zurückgegeben. Danach wird das BASIC-Programm fortgesetzt.

VAL {V<Shift>+A} — Funktion

```
A = VAL (String-Ausdruck)
```

VAL wandelt eine in einer Zeichenkette (in Form einer Zeichenfolge) enthaltene Zahl in eine "numerische" Zahl um, um diese beispielsweise für Berechnungen heranziehen zu können. VAL bildet das Gegenstück zu der Funktion STR$.

VERIFY {V<Shift>+E} — Befehl

```
VERIFY Name,Geräteadresse
```

Falls Sie nach einer SAVE-Anweisung nicht ganz sicher sind, ob das Programm korrekt gespeichert wurde, können Sie dies mit VERIFY überprüfen lassen. VERIFY vergleicht das momentan im Rechner befindliche Programm mit dem unter dem angege-

benen Namen auf Kassette oder Diskette gespeicherten. Stimmen die beiden Programme nicht überein, so erscheint die Meldung VERIFY ERROR, ansonsten OK.

WAIT {W<Shift>+A} Befehl

```
WAIT Speicheradresse, num. Ausdruck [,num. Ausdruck]
```

WAIT verknüpft den Inhalt der angegebenen Speicherzelle (0 bis 65.535) mit dem dritten Parameter (falls vorhanden) durch ein logisches EXCLUSIV-ODER und anschließend mit dem zweiten Parameter durch ein logisches UND. Solange sich bei diesen Verknüpfungen der Wert "Null" ergibt, werden sie wiederholt, d.h., die Programmausführung wird so lange angehalten, bis sich der Inhalt der angesprochenen Speicherzelle verändert. WAIT ist eine Anweisung, die man nur äußerst selten benötigt.

Anhang A.2: Fehlermeldungen

Nachfolgend finden Sie eine vollständige Liste der BASIC-2.0-Fehlermeldungen, jeweils mit einer Beschreibung der Fehlerursachen und Tips zur Behebung des Fehlers.

BAD SUBSCRIPT ERROR

Der Indexwert einer Feldvariablen ist größer als in der DIM-Anweisung vereinbart, oder die Anzahl der verwendeten Indizes (bei mehrdimensionalen Feldern) ist falsch.

BREAK ERROR

Während eines Lade- oder Speichervorgangs (mittels LOAD bzw. SAVE) wurde die <Stop>-Taste gedrückt.

CAN'T CONTINUE ERROR

Sie haben versucht, ein zuvor unterbrochenes BASIC-Programm mittels CONT fortzusetzen. Falls Sie vor CONT irgendwelche Veränderungen am Programm vorgenommen haben, beispielsweise durch Eingabe einer neuen Programmzeile, so bekommen Sie diese Fehlermeldung.

DEVICE NOT PRESENT ERROR

Das durch einen Ein-/Ausgabebefehl (INPUT#, PRINT# usw.) angesprochene Peripheriegerät (Datasette, Floppy, Drucker, usw.) ist nicht betriebsbereit. Vielleicht haben Sie auch die Geräteadresse verwechselt (z.B. für die Floppy eine "9" statt einer "8" geschrieben). Ein Tippfehler, der einem häufig passiert!

DIVISION BY ZERO ERROR

Es wurde versucht, durch "Null" zu dividieren, was mathematisch nicht erlaubt ist. Dieser Fehler tritt vor allem bei komplexeren Ausdrücken, wie etwa A/(B*C*D*E), auf. Bei diesem Ausdruck genügt es, wenn eine der Variablen B, C, D oder E "null" wird. Oft wurde auch einfach vergessen, der betreffenden Variablen einen Wert zuzuweisen.

EXTRA IGNORED ERROR

Sie haben bei einer INPUT-Abfrage mehr Eingaben getätigt als Variablen hinter dem INPUT-Befehl stehen. Die überzähligen Eingaben werden ignoriert, das Programm jedoch fortgesetzt! Dieser Fehler wird häufig durch Texteingaben verursacht, die ein Komma oder einen Doppelpunkt enthalten. Diese beiden Zeichen werden von BASIC nämlich als Trennzeichen interpretiert. "BEISPIEL: C64" zum Beispiel wertet der Rechner als "BEISPIEL" und "C64". Durch ein vorangestelltes Anführungszeichen (im Beispiel: ""BEISPIEL: C64") läßt sich dieses Problem umgehen.

FILE DATA ERROR

Es wurde versucht, aus einer Datei Zeichenketten in eine numerische Variable einzulesen.

FILE NOT FOUND ERROR

Die in einem Floppy-Befehl (z.B. OPEN) angegebene Datei existiert auf der momentan eingelegten Diskette nicht. Oft wurde einfach der Name falsch geschrieben.

FILE NOT OPEN ERROR

Eine durch einen Ein-/Ausgabebefehl angesprochene Datei wurde noch nicht (mittels OPEN) geöffnet (oder durch CLOSE bereits wieder geschlossen).

FILE OPEN ERROR

Es wurde versucht, eine Datei mit OPEN zu öffnen, die bereits geöffnet war.

FORMULA TOO COMPLEX ERROR

Dieser Fehler wird durch zu komplexe String-Ausdrücke verursacht, wie zum Beispiel ""A"+("BC"+("DE"+"EF"))". Ein Zeichenkettenausdruck darf nicht mehr als zwei Teilausdrücke in Klammern enthalten.

ILLEGAL DEVICE NUMBER ERROR

Diese Fehlermeldung zeigt an, daß ein Befehl an ein unzulässiges Gerät gesendet wurde, beispielsweise SAVE oder LOAD zur Tastatur (Geräteadresse 0). Wahrscheinlich haben Sie sich bei der Geräteadresse vertippt.

ILLEGAL DIRECT ERROR

Es wurde versucht, einen im Direktmodus unzulässigen Befehl (unter anderem INPUT und GET) dort zu verwenden.

ILLEGAL QUANTITY ERROR

Ein mathematischer Ausdruck hat einen Wert außerhalb des erlaubten Bereichs ergeben. Diese Fehlermeldung kann sehr viele Ursachen haben, etwa ein negativer Wert bei einem POKE-Befehl, eine Integer-Zahl kleiner als -32.768 oder eine logische File-Nummer größer als 255.

LOAD ERROR

Beim Laden eines Programms von Datasette oder Floppy trat ein Fehler auf.

MISSING FILENAME ERROR

Beim LOAD- oder SAVE-Befehl wurde vergessen, einen Dateinamen anzugeben.

NEXT WITHOUT FOR ERROR

Der BASIC-Interpreter traf auf ein NEXT, ohne zuvor eine entsprechende FOR-Anweisung abgearbeitet zu haben. Dieser Fehler tritt häufig in (evtl. zu stark) verschachtelten Schleifen auf.

NOT INPUT FILE ERROR

Es wurde versucht, aus einer nur zum Schreiben geöffneten Datei Daten zu lesen.

NOT OUTPUT FILE ERROR

Es wurde versucht, in eine nur zum Lesen geöffnete Datei Daten zu schreiben.

OUT OF DATA ERROR

In einem Programm wurde ein READ-Befehl angetroffen, ohne daß noch DATA-Werte vorhanden sind. Häufige Ursachen: Es wurde vergessen, den Datazeiger mittels RESTORE zurückzustellen. Zwischen zwei DATA-Werten fehlt das trennende Komma.

OUT OF MEMORY ERROR

Diese Fehlermeldung zeigt an, daß nicht mehr genügend BASIC- oder Stack-Speicherplatz vorhanden ist. In den meisten Fällen wurden entweder Variablenfelder zu groß dimensioniert oder Schleifen und Unterprogrammaufrufe zu tief verschachtelt. Der Fehler kann allerdings auch beim Nachladen von Maschinenprogrammen auftreten. In allen Fällen empfiehlt sich die Eingabe von NEW, um schlimmeres (wie etwa einen Rechnerabsturz) zu vermeiden.

OVERFLOW ERROR

Bei einer Fließkomma-Berechnung liegt das Endergebnis oder ein Zwischenergebnis außerhalb des zulässigen Wertebereichs.

REDIM'D ARRAY ERROR

Es wurde versucht, ein bereits existierendes Variablenfeld noch einmal zu dimensionieren. Innerhalb eines Programms darf ein Feld nur einmal (mit DIM) dimensioniert werden! Zu beachten ist dabei auch, daß Felder beim ersten Auftauchen eines Feldelementes innerhalb des Programms automatisch auf 11 Elemente vordimensioniert werden, sofern sie nicht bereits dimensioniert wurden.

REDO FROM START ERROR

Bei einer INPUT-Abfrage wurde statt einer Zahl eine Zeichenkette eingegeben. Die INPUT-Abfrage wird deshalb wiederholt, das Programm also nicht abgebrochen!

RETURN WITHOUT GOSUB ERROR

Der BASIC-Interpreter traf auf ein RETURN, ohne zuvor eine GOSUB-Anweisung abgearbeitet zu haben. Häufige Ursache: Am Ende des Hauptprogramms (vor den Unterprogrammen) wurde vergessen, eine END-Anweisung einzufügen.

STRING TOO LONG ERROR

Es wurde versucht, einer String-Variablen mehr als die erlaubten 255 Zeichen (bzw. beim INPUT#-Befehl mehr als 88 Zeichen) zuzuweisen.

SYNTAX ERROR

Diese Fehlermeldung zeigt ganz allgemein an, daß eine Anweisung von BASIC nicht richtig interpretiert werden konnte. Die häufigste Ursache dafür dürften Tippfehler sein. Erscheint die Meldung nach NEW oder RUN, so ist das erste Byte des BASIC-Speichers ungleich Null. "POKE (PEEK(43)+PEEK(44)*256),0" sorgt in diesem Fall für Abhilfe.

TOO MANY FILES ERROR

Es wurde versucht, mittels OPEN mehr als die erlaubten zehn logischen Dateien gleichzeitig zu öffnen.

TYPE MISMATCH ERROR

Es wurde versucht, einer numerischen Variablen einen String-Ausdruck zuzuweisen oder umgekehrt.

UNDEF'D FUNCTION ERROR

In einem numerischen Ausdruck wurde eine Funktion angesprochen, ohne zuvor mit DEF FN definiert zu werden. Vielleicht wurde nur der Name der Funktion falsch geschrieben.

UNDEF'D STATEMENT ERROR

Über RUN, GOTO oder GOSUB wurde versucht, eine im Programm nicht existierende Programmzeile anzuspringen.

VERIFY ERROR

Diese Fehlermeldung zeigt an, daß das im Rechnerspeicher befindliche Programm mit dem bei VERIFY angegebenen (und auf Disk oder Kasette gespeicherten) Programm nicht identisch ist, der Speichervorgang also fehlerhaft verlief. Das Programm sollte auf einer anderen Diskette oder Kasette gespeichert werden.

Anhang B: Assembler-Referenz

Dieser Anhang enthält eine komplette Übersicht über den Assembler-Befehlssatz des Commodore 64. Die Sachgruppen-Kurzübersicht dient zum schnellen Ermitteln eines bestimmten Befehls. Ausführlichere Informationen zu allen Befehlen finden Sie dann in der alphabetischen Übersicht.

Neben der Wirkung des Befehls und aller möglichen Adressierungsarten erfahren Sie dort den Code und die Länge des Befehls, welche Prozessor-Flags er beeinflußt sowie die zu seiner Abarbeitung erforderlichen Taktzyklen.

Anhang B.1: Sachgruppen-Kurzübersicht

Vergleichsbefehle

CMP	Vergleicht Speicherzelle mit Akku.
CPX	Vergleicht Speicherzelle mit X-Register.
CPY	Vergleicht Speicherzelle mit Y-Register.

Sprungbefehle

BCC	Verzweigt, falls das Carry-Flag gelöscht ist.
BCS	Verzweigt, falls das Carry-Flag gesetzt ist.
BEQ	Verzweigt, falls das Zero-Flag gelöscht ist.
BNE	Verzweigt, falls das Zero-Flag gesetzt ist.
BMI	Verzweigt, falls das Negativ-Flag gesetzt ist.
BPL	Verzweigt, falls das Negativ-Flag gelöscht ist.
B	Verzweigt, falls das Overflow-Flag gelöscht ist.
BVS	Verzweigt, falls das Overflow-Flag gesetzt ist.
JMP	Verzweigt zur angegebenen Adresse.
JSR	Verzweigt zur angegebenen Adresse (Unterprogrammaufruf).
NOP	Keine Operation.
RTS	Rücksprung aus Unterprogramm.

Unterbrechungsbefehle

BRK Programmunterbrechung.
CLI Interrupt-Flag löschen.
RTI Rücksprung aus Interrupt.
SEI Interrupt-Flag setzen.

Transferbefehle

LDA Lädt Akku mit angegebenem Wert.
LDX Lädt X-Register mit angegebenem Wert.
LDY Lädt Y-Register mit angegebenem Wert.
PHA Bringt Akku-Inhalt auf Stack.
PHP Bringt Status-Register auf Stack.
PLA Lädt Akku mit oberstem Stack-Element.
PLP Lädt Status-Register mit oberstem Stack-Element.
STA Schreibt Akku-Inhalt in angegebene Speicherzelle.
STX Schreibt X-Register in angegebene Speicherzelle.
STY Schreibt Y-Register in angegebene Speicherzelle.
TAX Schreibt Akku-Inhalt ins X-Register.
TAY Schreibt Akku-Inhalt ins Y-Register.
TXA Schreibt X-Register-Inhalt in den Akku.
TYA Schreibt Y-Register-Inhalt in den Akku.
TSX Schreibt Stackpointer-Inhalt ins X-Register.
TXS Schreibt X-Register-Inhalt in den Stackpointer.

In-/Dekrementierbefehle

DEC Dekrementiert den Inhalt der angegebenen Speicherzelle.
DEX Dekrementiert den Inhalt des X-Registers.
DEY Dekrementiert den Inhalt des Y-Registers.
INC Inkrementiert den Inhalt der angegebenen Speicherzelle.
INX Inkrementiert den Inhalt des X-Registers.
INY Inkrementiert den Inhalt des Y-Registers.

Arithmetikbefehle

ADC Addiert Operand + Carry-Flag zum Akku-Inhalt.
CLC Carry-Flag löschen.
CLD Dezimal-Flag löschen.
CLV Overflow-Flag löschen.

SBC Subtrahiert Operand + Carry-Flag vom Akku-Inhalt.
SEC Carry-Flag setzen.
SED Dezimal-Flag setzen.

Logische Befehle

AND Verknüpft Akku-Inhalt mit Operand durch logisches UND.
BIT Verknüpft Akku-Inhalt mit Operand durch logisches UND und setzt die entsprechenden Flags des Status-Registers.
EOR Verknüpft Akku-Inhalt mit Operand durch logisches EXCLUSIV-ODER.
ORA Verknüpft Akku-Inhalt mit Operand durch logisches ODER.

Schiebe- und Rotierbefehle

ASL Verschiebt die Bits des Operanden um eine Stelle nach links.
LSR Verschiebt die Bits des Operanden um eine Stelle nach rechts.
ROL Rotiert die Bits des Operanden um eine Stelle nach links.
ROR Rotiert die Bits des Operanden um eine Stelle nach rechts.

Anhang B.2: Alphabetische Übersicht

ADC — Add with carry

Addiert Operand + Carry-Flag zum Akku-Inhalt.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x	x				x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	ADC #Op	$69	2	2
Absolut	ADC Op	$6D	3	4
Zeropage	ADC Op	$65	2	3
Abs.-X-Indiziert	ADC Op,X	$7D	3	4
Abs.-Y-Indiziert	ADC Op,Y	$79	3	4
Zerop.-X-Indiziert	ADC Op,X	$75	2	4
Indirekt-X-Indiziert	ADC (Op,X)	$61	2	6
Indirekt-Y-Indiziert	ADC (Op),Y	$71	2	5

AND — AND Accu

Verknüpft Akku-Inhalt mit Operand durch logisches UND.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	AND #Op	$29	2	2
Absolut	AND Op	$2D	3	4
Zeropage	AND Op	$25	2	3
Abs.-X-Indiziert	AND Op,X	$3D	3	4
Abs.-Y-Indiziert	AND Op,Y	$39	3	4
Zerop.-X-Indiz.	AND Op,X	$35	2	4
Indirekt-X-Ind.	AND (Op,X)	$21	2	6
Indirekt-Y-Ind.	AND (Op),Y	$31	2	5

ASL — Arithmetic shift left

Verschiebt die Bits des Operanden um eine Stelle nach links.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Akku	ASL	$0A	1	2
Absolut	ASL Op	$0E	3	6
Zeropage	ASL Op	$06	2	5
Abs.-X-Indiziert	ASL Op,X	$1E	3	7
Zerop.-X-Indiz.	ASL Op,X	$16	2	6

BCC — Branch if carry clear

Verzweigt, falls das Carry-Flag gelöscht ist.

Beeinflußte Flags: N V B D I Z C
(Keine)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Relativ	BCC Op	$90	2	2

BCS — Branch if carry set

Verzweigt, falls das Carry-Flag gesetzt ist.

Beeinflußte Flags: N V B D I Z C
(Keine)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Relativ	BCS Op	$B0	2	2

BEQ — Branch if equal to zero

Verzweigt, falls das Zero-Flag gelöscht ist.

Beeinflußte Flags: N V B D I Z C
(Keine)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Relativ	BEQ Op	$F0	2	2

BIT — Test bits

Verknüpft Akku-Inhalt mit Operand durch logisches UND und setzt die entsprechenden Flags des Status-Registers.

```
Beeinflußte Flags:  N  V  B  D  I  Z  C
                    x  x           x
Adressierungsart  Schreibweise  Code  Länge  Taktzyklen
Absolut           BIT Op        $2C    3        4
Zeropage          BIT Op        $24    2        3
```

BNE — Branch if not equal to zero

Verzweigt, falls das Zero-Flag gesetzt ist.

```
Beeinflußte Flags:  N  V  B  D  I  Z  C
                    (Keine)
Adressierungsart  Schreibweise  Code  Länge  Taktzyklen
Relativ           BNE Op        $D0    2        2
```

BMI — Branch if minus

Verzweigt, falls das Negativ-Flag gesetzt ist.

```
Beeinflußte Flags:  N  V  B  D  I  Z  C
                    (Keine)
Adressierungsart  Schreibweise  Code  Länge  Taktzyklen
Relativ           BMI Op        $30    2        2
```

BPL — Branch if plus

Verzweigt, falls das Negativ-Flag gelöscht ist.

```
Beeinflußte Flags:  N  V  B  D  I  Z  C
                    (Keine)
Adressierungsart  Schreibweise  Code  Länge  Taktzyklen
Relativ           BPL Op        $10    2        2
```

BRK — Break

Programmunterbrechung.

Beeinflußte Flags:	N	V	B	D	I	Z	C
			x				

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	BRK	$00	1	7

B — Branch if overflow clear

Verzweigt, falls das Overflow-Flag gelöscht ist.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	(Keine)						

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Relativ	BOp	$50	2	2

BVS — Branch if overflow set

Verzweigt, falls das Overflow-Flag gesetzt ist.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	(Keine)						

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Relativ	BVS Op	$70	2	2

CLC — Clear carry

Carry-Flag löschen.

Beeinflußte Flags:	N	V	B	D	I	Z	C
							x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	CLC	$18	1	2

CLD — Clear decimal mode

Dezimal-Flag löschen.

Beeinflußte Flags:	N	V	B	D	I	Z	C
				x			

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	CLD	$D8	1	2

CLI — Clear interrupt flag

Interrupt-Flag löschen.

Beeinflußte Flags:	N	V	B	D	I	Z	C
					x		

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	CLI	$58	1	2

CLV — Clear overflow flag

Overflow-Flag löschen.

Beeinflußte Flags:	N	V	B	D	I	Z	C
		x					

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	CLV	$B8	1	2

CMP — Compare with accu

Vergleicht Speicherzelle mit Akku.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	CMP #Op	$C9	2	2
Absolut	CMP Op	$CD	3	4
Zeropage	CMP Op	$C5	2	3

Abs.-X-Indiziert	CMP Op,X	$DD	3	4
Abs.-Y-Indiziert	CMP Op,Y	$D9	3	4
Zerop.-X-Indiz.	CMP Op,X	$D5	2	4
Indirekt-X-Ind.	CMP (Op,X)	$C1	2	6
Indirekt-Y-Ind.	CMP (Op),Y	$D1	2	5

CPX — Compare with x-register

Vergleicht Speicherzelle mit X-Register.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	CPX #Op	$E0	2	2
Absolut	CPX Op	$EC	3	4
Zeropage	CPX Op	$E4	2	3

CPY — Compare with y-register

Vergleicht Speicherzelle mit Y-Register.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	CPY #Op	$C0	2	2
Absolut	CPY Op	$CC	3	4
Zeropage	CPY Op	$C4	2	3

DEC — Decrement memory

Dekrementiert den Inhalt der angegebenen Speicherzelle.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Absolut	DEC Op	$CE	3	6
Zeropage	DEC Op	$C6	2	5
Abs.-X-Indiziert	DEC Op,X	$DE	3	7
Zerop.-X-Indiz.	DEC Op,X	$D6	2	6

DEX — Decrement x-register

Dekrementiert den Inhalt des X-Registers.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	DEX	$CA	1	2

DEY — Decrement y-register

Dekrementiert den Inhalt des Y-Registers.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	DEY	$88	1	2

EOR — exclusive-or accu

Verknüpft Akku-Inhalt mit Operand durch logisches EXCLUSIV-ODER.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	EOR #Op	$49	2	2
Absolut	EOR Op	$4D	3	4
Zeropage	EOR Op	$45	2	3
Abs.-X-Indiziert	EOR Op,X	$5D	3	4
Abs.-Y-Indiziert	EOR Op,Y	$59	3	4
Zerop.-X-Indiz.	EOR Op,X	$55	2	4
Indirekt-X-Ind.	EOR (Op,X)	$41	2	6
Indirekt-Y-Ind.	EOR (Op),Y	$51	2	5

INC — Increment memory

Inkrementiert den Inhalt der angegebenen Speicherzelle.

Beeinflußte Flags: N V B D I Z C
x (N), x (Z)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Absolut	INC Op	$EE	3	6
Zeropage	INC Op	$E6	2	5
Abs.-X-Indiziert	INC Op,X	$FE	3	7
Zerop.-X-Indiz.	INC Op,X	$F6	2	6

INX — Increment x-register

Inkrementiert den Inhalt des X-Registers.

Beeinflußte Flags: N V B D I Z C
x (N), x (Z)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	INX	$E8	1	2

INY — Increment y-register

Inkrementiert den Inhalt des Y-Registers.

Beeinflußte Flags: N V B D I Z C
x (N), x (Z)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	INY	$C8	1	2

JMP — Jump

Verzweigt zur angegebenen Adresse.

Beeinflußte Flags: N V B D I Z C
(Keine)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Indirekt	JMP (Op)	$6C	3	5
Absolut	JMP Op	$4C	3	3

JSR — Jump to subroutine

Verzweigt zur angegebenen Adresse (Unterprogrammaufruf).

Beeinflußte Flags:	N	V	B	D	I	Z	C
	(Keine)						

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Absolut	JSR Op	$20	3	6

LDA — Load accu

Lädt Akku mit angegebenem Wert.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	LDA #Op	$A9	2	2
Absolut	LDA Op	$AD	3	4
Zeropage	LDA Op	$A5	2	3
Abs.-X-Indiziert	LDA Op,X	$BD	3	4
Abs.-Y-Indiziert	LDA Op,Y	$B9	3	4
Zerop.-X-Indiz.	LDA Op,X	$B5	2	4
Indirekt-X-Ind.	LDA (Op,X)	$A1	2	6
Indirekt-Y-Ind.	LDA (Op),Y	$B1	2	5

LDX — Load x-register

Lädt X-Register mit angegebenem Wert.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	LDX #Op	$A2	2	2
Absolut	LDX Op	$AE	3	4
Zeropage	LDX Op	$A6	2	3
Abs.-Y-Indiziert	LDX Op,Y	$BE	3	4
Zerop.-Y-Indiz.	LDX Op,Y	$B6	2	4

LDY — Load y-register

Lädt Y-Register mit angegebenem Wert.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	LDY #Op	$A0	2	2
Absolut	LDY Op	$AC	3	4
Zeropage	LDY Op	$A4	2	3
Abs.-X-Indiziert	LDY Op,X	$BC	3	4
Zerop.-X-Indiz.	LDY Op,X	$B4	4	2

LSR — Logical shift right

Verschiebt die Bits des Operanden um eine Stelle nach rechts.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Akku	LSR	$4A	1	2
Absolut	LSR Op	$4E	3	6
Zeropage	LSR Op	$46	2	5
Abs.-X-Indiziert	LSR Op,X	$5E	3	7
Zerop.-X-Indiz.	LSR Op,X	$56	2	6

NOP — No operation

Keine Operation.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	(Keine)						

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	NOP	$EA	1	2

ORA — OR accu

Verknüpft Akku-Inhalt mit Operand durch logisches ODER.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	ORA #Op	$09	2	2
Absolut	ORA Op	$0D	3	4
Zeropage	ORA Op	$05	2	3
Abs.-X-Indiziert	ORA Op,X	$1D	3	4
Abs.-Y-Indiziert	ORA Op,Y	$19	3	4
Zerop.-X-Indiz.	ORA Op,X	$15	2	4
Indirekt-X-Ind.	ORA (Op,X)	$01	2	6
Indirekt-Y-Ind.	ORA (Op),Y	$11	2	5

PHA — Push accu

Bringt Akku-Inhalt auf Stack.

Beeinflußte Flags: N V B D I Z C
(Keine)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	PHA	$48	1	3

PHP — Push processor-status

Bringt Status-Register auf Stack.

Beeinflußte Flags: N V B D I Z C
(Keine)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	PHP	$08	1	3

PLA — Pull accu

Lädt Akku mit oberstem Stack-Element.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	PLA	$68	1	3

PLP — Pull processor-status

Lädt Status-Register mit oberstem Stack-Element.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x	x	x	x	x	x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	PLP	$28	1	4

ROL — Rotate left

Rotiert die Bits des Operanden um eine Stelle nach links.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Akku	ROL	$2A	1	2
Absolut	ROL Op	$2E	3	6
Zeropage	ROL Op	$26	2	5
Abs.-X-Indiziert	ROL Op,X	$3E	3	7
Zerop.-X-Indiz.	ROL Op,X	$36	2	6

ROR — Rotate right

Rotiert die Bits des Operanden um eine Stelle nach rechts.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Akku	ROR	$6A	1	2
Absolut	ROR Op	$6E	3	6
Zeropage	ROR Op	$66	2	5
Abs.-X-Indiziert	ROR Op,X	$7E	3	7
Zerop.-X-Indiz.	ROR Op,X	$76	2	6

RTI — Return from interrupt

Rücksprung aus Interrupt.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x	x	x	x	x	x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	RTI	$40	1	6

RTS — Return from subroutine

Rücksprung aus Unterprogramm.

Beeinflußte Flags: N V B D I Z C
(Keine)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	RTS	$60	1	6

SBC — Substract with carry

Subtrahiert Operand + Carry-Flag vom Akku-Inhalt.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x	x				x	x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Unmittelbar	SBC #Op	$E9	2	2
Absolut	SBC Op	$ED	3	4
Zeropage	SBC Op	$E5	2	3
Abs.-X-Indiziert	SBC Op,X	$FD	3	4
Abs.-Y-Indiziert	SBC Op,Y	$F9	3	4
Zerop.-X-Indiz.	SBC Op,X	$F5	2	4
Indirekt-X-Ind.	SBC (Op,X)	$E1	2	6
Indirekt-Y-Ind.	SBC (Op),Y	$F1	2	5

SEC — Set carry

Carry-Flag setzen.

Beeinflußte Flags:	N	V	B	D	I	Z	C
							x

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	SEC	$38	1	2

SED — Set decimal mode

Dezimal-Flag setzen.

Beeinflußte Flags:	N	V	B	D	I	Z	C
				x			

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	SED	$F8	1	2

SEI — Set interrupt

Interrupt-Flag setzen.

Beeinflußte Flags:	N	V	B	D	I	Z	C
					x		

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	SEI	$78	1	2

STA — Store accu

Schreibt Akku-Inhalt in angegebene Speicherzelle.

Beeinflußte Flags: N V B D I Z C
(Keine)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Absolut	STA Op	$8D	3	4
Zeropage	STA Op	$85	2	3
Abs.-X-Indiziert	STA Op,X	$9D	3	5
Abs.-Y-Indiziert	STA Op,Y	$99	3	5
Zerop.-X-Indiz.	STA Op,X	$95	2	4
Indirekt-X-Ind.	STA (Op,X)	$81	2	6
Indirekt-Y-Ind.	STA (Op),Y	$91	2	6

STX — Store x-register

Schreibt X-Register in angegebene Speicherzelle.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	(Keine)						

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Absolut	STX Op	$8E	3	4
Zeropage	STX Op	$86	2	3
Zerop.-Y-Indiz.	STX Op,Y	$96	2	4

STY — Store y-register

Schreibt Y-Register in angegebene Speicherzelle.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	(Keine)						

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Absolut	STY Op	$8C	3	4
Zeropage	STY Op	$84	2	3
Zerop.-X-Indiz.	STY Op,X	$94	4	2

TAX — Transfer accu to x-register

Schreibt Akku-Inhalt ins X-Register.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	TAX	$AA	1	2

TAY — Transfer accu to y-register

Schreibt Akku-Inhalt ins Y-Register.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	TAY	$A8	1	2

TXA — Transfer x-register to accu

Schreibt X-Register-Inhalt in den Akku.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	TXA	$8A	1	2

TYA — Transfer y-register to accu

Schreibt Y-Register-Inhalt in den Akku.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	TYA	$98	1	2

TSX — Transfer stackregister to x-register

Schreibt Stackpointer-Inhalt ins X-Register.

Beeinflußte Flags:	N	V	B	D	I	Z	C
	x					x	

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	TSX	$BA	1	2

TXS — Transfer x-register to stackregister

Schreibt X-Register-Inhalt in den Stackpointer.

Beeinflußte Flags: N V B D I Z C
(Keine)

Adressierungsart	Schreibweise	Code	Länge	Taktzyklen
Implizit	TXS	$9A	1	2

Anhang C: Die GEOS-Sprungtabelle

Viellecht kennen Sie ja bereits die Sprungtabelle die Ihr C64 benutzt. Siehe im Anhang "Alphabetisches Verzeichnis aller ROM-Routinen".

Eine solche Sprungtabelle gibt es auch in GEOS. Sie liegt im Speicher ab $C100. Über diese Tabelle können alle wichtigen Aktionen des GEOS-Kernal benutzt werden. Zusätzlich besteht eine große Wahrscheinlichkeit, daß bei neuen Versionen des GEOS-Kernal die Routinen der Sprungtabelle problemlos weiterbenutzt werden können. So haben sich beispielsweise von GEOS V1.1 über V1.2 zu GEOS V1.3 viele Routinen geändert oder liegen an anderen Adressen. Die Tabelleneinträge zeigen aber immer noch auf die gleichen Routinen.

Auf den folgenden Seiten finden Sie nicht nur die Sprungtabelle abgedruckt, sondern auch die Funktion und Übergabeparameter der Routinen. Wenn Sie sich einen disassemblierten Ausdruck des GEOS-Kernal erstellen, können Sie sich diese Routinen genau anschauen und damit problemlos arbeiten. Halten Sie sich bei eigenen Programmen unbedingt an die Sprungtabelle, damit Ihr Programm auch bei neueren Versionen von GEOS voll funktionstüchtig bleibt.

C100	InterruptMain

Hier steht ein Sprung zum GEOS-IRQ. Normalerweise erledigt GEOS die IRQ-Abarbeitung von selbst. Ein solcher IRQ findet alle 1/50 Sekunde statt (Rasterzeilen-Interrupt). Dabei werden die folgenden Arbeiten erledigt:

1. Maus bewegen und deren Bereich testen.
2. Timergruppe A dekrementieren (Multi-Tasking-Jobs).
3. Timergruppe B dekrementieren (Delay-Jobs).
4. Cursor bedienen, falls das 7. Bit in $84B4 gesetzt ist ("Alpha-Flag").

5. In $850A (Low) und $850B (High) eine Zufallszahl erzeugen. Diese Werte können durch JSR GetRandom ausgelesen werden (GetRandom=$C187).

C103	InitProcesses

Jobs vom Typ A (s. InterruptMain) in Job-Puffer 1 schreiben.

Akku	Anzahl der Jobs (maximal 20!)
$02,$03	Adresse der Tabelle

Diese Tabelle muß folgenden Aufbau haben:

```
Job-Adresse (Job 1)  Low
Job-Adresse          High
Timer-Wert           Low
Timer-Wert           High
Job-Adresse (Job 2)  Low
Job-Adresse          High
Timer-Wert           Low
Timer-Wert           High
```

und so weiter. In diese Tabelle trägt man Jobs ein, die immer wieder ausgeführt werden sollen. Der zugehörige Timer-Wert entscheidet über die Häufigkeit der Ausführung. Dazu wird vom GEOS-Kernal jedem Job ein Status-Byte zugeordnet, mit dem Jobs aktiviert und inaktiviert werden können. Der IRQ zählt die zugehörigen Timer-Werte (sie bilden die Timer-Gruppe "A", siehe $C100) auf "0" herunter und gibt den Job dann zur Abarbeitung durch die GEOS-Main-Loop (siehe $C1C3) frei. Somit können rein rechnerisch Timer-Werte zwischen 1/50 Sekunde (Timer-Wert=$0001) und ca. 21 Stunden, 50 Minuten (Timer-Wert=$FFFF) programmiert werden. Die einzelnen Bits im Status-Byte haben folgende Bedeutung:

Bit 5	Wenn dieses Bit gesetzt ist, zählt der IRQ den zugehörigen Timer nicht weiter herunter. Der Job ist damit auf dem augenblicklichen Stand eingefroren (FREEZE).

Bit 6 Verhindert, daß der Job ausgeführt wird, auch wenn der Timer-Wert abgelaufen ist (BLOCK).

Bit 7 Wird vom IRQ als Zeichen dafür gesetzt, daß der zugehörige Timer abgelaufen ist. Dadurch kann der Job von der Hauptschleife abgearbeitet werden. Diese setzt sofort Bit 7 zurück (Disable) und startet den zugehörigen Prozeß. Durch das Zurücksetzen wird eine erneute Abarbeitung verhindert (siehe auch $C109=EnableProcess).

Durch InitProcesses werden die Jobs nur eingetragen. Da aber hierbei Bit 5 (FREEZE) gesetzt wird, werden vom IRQ noch nicht die Timer gezählt. Dazu muß jeder Job erst zur Bearbeitung freigegeben werden (RestartProcess=$C106). Wichtig ist, daß man diese Jobs alle auf einmal eintragen muß. Ein nachträgliches Zufügen von Jobs ist nicht vorgesehen. Damit wird diese Routine immer zu dem Initialisierungsteil eines Programms gehören.

C106 RestartProcess

Löscht Bit 5 und Bit 6 im Status-Byte und trägt die Timer-Werte ein. Dadurch wird jede Blockierung aufgehoben (UNFREEZE und UNBLOCK). Die zugehörige Job-Nummer wird im X-Register übergeben.

C109 EnableProcess

Obwohl die Jobs normalerweise durch heruntergezählte Timer gestartet werden, kann jeder Job durch EnableProcess auch "von Hand" gestartet werden. Der Timer-Wert des durch das X-Register bezeichneten Jobs wird sofort auf Null gesetzt. Beim nächsten Durchlauf der Hauptschleife wird dann dieser Job ausgeführt.

C10C	BlockProcess

Setzt Bit 6 im Status-Byte des im X-Register übergebenen Jobs. Dadurch werden zwar die Timer-Werte weiter dekrementiert, dieser Job aber bei "0" dennoch nicht ausgeführt.

C10F	UnblockProcess

Gegenstücke zu BlockProcess. Job-Blockierung durch Löschen von Bit 6 im Status-Byte aufheben. Die Job-Nummer muß im X-Register übergeben werden. Mit der Länge des blockierten Zeitraumes wächst die Wahrscheinlichkeit, daß inzwischen die Timer-Werte vom IRQ heruntergezählt worden sind. In diesem Fall wird der Job beim nächsten Durchlauf der MainLoop abgearbeitet.

C112	FreezeProcess

Durch das Setzen des Bit 5 wird der Prozeß mit seinem aktuellen Timer-Stand eingefroren. Das X-Register muß die Job-Nummer enthalten.

C115	UnfreezeProzess

Löscht Bit 5 im Status-Byte des im X-Register übergebenen Jobs. Dadurch werden die vorher gestoppten Timer-Werte weitergezählt. Auch beim Aufruf dieser Routine muß das X-Register die Job-Nummer enthalten.

Grafikbibliothek

Bei der Benutzung der Grafikroutinen hat das Byte $2F entscheidend Einfluß darauf, wohin Informationen geschrieben oder von wo sie geholt werden. Abhängig von $2F werden nämlich die Zeiger auf die einzelnen Bildschirme gesetzt. Folgende Werte für $2F sind vorgesehen:

$80	Ausgabe nur auf sichtbaren Bildschirm ($A000)
$40	Ausgabe nur auf unsichtbaren Bildschirm ($6000)
$C0	Ausgabe auf beide Bildschirme ($C0=$80 OR $40)

C118	HorizontalLine

Zieht eine waagerechte Linie auf dem hochauflösenden Grafikbildschirm (vorher in $2F den Bildschirm auswählen!).

Parameter

Akku	Die Bits bestimmen direkt das Strichmuster
$08,$09	Linke Grenze (Low,High)
$0A,$0B	Rechte Grenze (Low,High)
$18	Y-Position der Linie
X-Werte	(0-319)
Y-Wert	(0-199)

C11B	InvertLine

Invertiert eine waagerechte Linie auf dem Grafikbildschirm (vorher in $2F den Bildschirm auswählen!).

Parameter

$08,$09	Linke Grenze (Low,High)
$0A,$0B	Rechte Grenze (Low,High)
$18	Y-Position der Linie

X-Wert	(0-319)
Y-Wert	(0-199)

C11E **RecoverLine**

Holt eine Linie aus dem zweiten Bildschirm (ab $6000) auf den sichtbaren Bildschirm zurück. Wird beispielsweise benutzt, um beim Abbau von Menüs den ursprünglichen Bildschirm wiederherzustellen.

$08,$09	Linke Grenze (Low,High)
$0A,$0B	Rechte Grenze (Low,High)
$18	Y-Position der Linie

C121 **VerticalLine**

Zieht eine senkrechte Linie (vorher in $2F den Bildschirm auswählen!).

Akku	Muster der Linie
$08	O+berere Grenze
$09	Untere Grenze
$0A	Position Low
$0B	X-Position High

C124 **Rectangle**

Füllt ein Rechteck mit einem Muster (vorher in $2F den Bildschirm auswählen!).

$06	Obere Grenze
$07	Untere Grenze

$08,$09	Linke Grenze (Low,High)
$0A,$0B	Rechte Grenze (Low,High)
$22,23	Zeiger auf das Muster

Das Muster muß aus 8 Bytes bestehen und somit eine 8x8-Punkte-Matrix ergeben. Durch vorherigen Aufruf der Routine "SetPattern" ($C139) kann auf einfache Art eins der 34 fertigen Muster des Kernal vorgegeben werden. Dazu muß SetPattern im Akku die Nummer des gewünschten Musters (0-33) übergeben werden.

C127 FrameRectangle

Malt einen rechteckigen Rahmen (vorher in $2F den Bildschirm auswählen!).

Akku	Die Bits des Akkus ergeben das Muster.
$06	Obere Grenze
$07	Untere Grenze
$08,$09	Linke Grenze (Low,High)
$0A,$0B	Rechte Grenze (Low,High)

C12A InvertRectangle

Invertiert ein Rechteck (vorher in $2F Bildschirm auswählen!).

$06	Obere Grenze
$07	Untere Grenze
$08,$09	Linke Grenze (Low,High)
$0A,$0B	Rechte Grenze (Low,High)

C12D	RecoverRectangle

Holt ein Rechteck aus dem zweiten Bildschirm zurück. Wird beispielsweise benutzt, um beim Abbau von Fenstern den ursprünglichen Bildschirminhalt wiederherzustellen.

$06	Obere Grenze
$07	Untere Grenze
$08,$09	Linke Grenze (Low,High)
$0A,$0B	Rechte Grenze (Low,High)

C130	DrawLine

Diese Routine zieht eine beliebig orientierbare Linie. Sie ist daher wesentlich flexibler als "HorizontalLine" oder "VerticalLine", aber auch erheblich langsamer. Vor ihrem Aufruf muß in $2F der Bildschirm ausgewählt werden.

$08	X-Position Punkt 1 Low
$09	X-Position Punkt 1 High
$0A	X-Position Punkt 2 Low
$0B	X-Position Punkt 2 High
$18	Y-Position Punkt 1
$19	Y-Position Punkt 2

Zusätzlich bestimmen folgende Status-Bits beim Einsprung die Ausführung dieser Funktion:

Bit 7 (Negativ-Flag) = 1

Dann ist Bit 0 (Carry-Flag) ohne Bedeutung. In diesem Fall wird die Linie vom zweiten, unsichtbaren Bildschirm auf den sichtbaren Bildschirm kopiert.

Mit dieser Funktion kann GEOPAINT ein UNDO realisieren. Die aktuelle Aktion erfolgt nur auf dem sichtbaren Bildschirm.

Soll sie widerrufen werden, wird dieselbe Funktion noch einmal in diesem Modus (Bit 7=1) aufgerufen. Dadurch wird genau der veränderte Teil des Bildschirms vom zweiten Bildschirm geholt und in den sichtbaren kopiert.

Bit 7 (Negativ-Flag)=0

In diesem Fall entscheidet das Carry-Flag folgendermaßen:

```
Carry-Flag=1 zieht eine Linie
Carry-Flag=0 löscht eine Linie
```

Bei diesem Modus muß wieder der gewünschte Bildschirm in $2F eingetragen werden.

C133	**DrawPoint**

Diese Routine zeichnet einen Punkt oder kopiert ihn vom Hintergrundbildschirm auf den sichtbaren Bildschirm. Es gelten die gleichen Bedingungen wie bei "DrawLine" (Status-Bits, $2F).

$08	X-Position Punkt Low
$09	X-Position Punkt High
$18	Y-Position Punkt Low
$19	Y-Position Punkt High

C136	**GraphicsString**

Führt vorher definierte Grafik-Jobs aus einer Tabelle aus. In $02,$03 wird ein Zeiger auf eine Tabelle übergeben, die die gewünschten Job-Nummern enthält.

Hinter der Job-Nummer folgen jeweils direkt die nötigen Daten für den Job. Das Ende der Tabelle muß durch die Job-Nummer 0 gekennzeichnet werden.

Job-Nummer 0 = RTS = Ende

Job-Nummer 1 = GetStartPoint

Funktion: 3 Bytes holen und diese dann nach $87D4 (X-Low),$87D5(X-High),$87D6 (Y) schreiben. Diese Koordinaten werden bei den folgenden Jobs als Ausgangspunkt benutzt. Beispielsweise muß bei der folgenden Routine Job-Nummer 2 zuerst Job (1) aufgerufen werden, dann kann DrawLineToPoint benutzt werden.

Job-Nummer 2 = DrawLineToPoint

Funktion: Die vorher von Job 1 eingetragenen Koordinaten nach $08, $09, $18 übernehmen und Job 1 ausführen, um noch die Koordinaten eines weiteren Punktes zu holen (drei Bytes). Dann diese neuen 3 Bytes nach $0A, $0B, $19 schreiben. Anschließend wird DrawLine mit gesetztem Carry- und gelöschtem V-Flag aufgerufen (siehe $C130).

Job-Nummer 3 = Rectangle

Funktion: Holt eine zweite Koordinate (3 Bytes). Berechnet aus den beiden Punkten ein Rechteck und füllt dieses mit dem derzeit aktuellen Muster.

Voraussetzung für den Aufruf von Job 3: Mit Job 1 die Koordinaten des ersten Eckpunktes festlegen und mit Job 5 das gewünschte Muster auswählen. Dann kann FillArea aufgerufen werden. Beispiel für eine Tabelle, die mit Job 3 ein Rechteck von P1 zu P2 mit einem Muster füllt:

```
$03,02 = Zeiger auf Tabelle = $1000
```

Anschließend: JSR GraphicsString

Daten in der Tabelle:

```
$1000 - $01  GetStartPoint (P1)
$1001 - $0A  X-Low von P1
$1002 - $00  X-High von P1
$1003 - $10  Y von P1
$1004 - $05  SetPattern
```

```
$1005 - $01  Musternummer (schwarz)
$1006 - $03  FillArea
$1007 - $50  X-Low von P2
$1008 - $00  X-High von P2
$1009 - $60  Y von P2
$100A - $00  Tabellenende
```

Job 4 = NOP

Dieser Job bewirkt nichts. Danach wird die Tabelle einfach weiter abgearbeitet.

Job 5 = SetPattern

Funktion: Das folgende Byte gibt die Nummer des gewünschten Musters an (0-33). Über das Aussehen der Muster können Sie sich bei GEOPAINT in der Musterleiste informieren.

Job 6 = PutString

Funktion: Gibt an einer bestimmten Position einen Text aus. Benötigt werden drei Bytes für die Position: X-Low, X-High, Y und ein folgender 2-Byte-Zeiger auf den Text. Der Text muß mit "0" abgeschlossen sein.

Job 7 = FrameRectangle

Funktion: Einen rechteckigen Rahmen zeichnen. Benötigt werden wieder die drei Bytes, die die Position des zweiten Punktes P2 angeben. P1 muß durch Job-Nummer 1 gesetzt sein.

Job 8 = XOffset

Funktion: Erhöht die X-Koordinaten von P1 um den angegebenen Wert. Benötigt werden die zwei Bytes X-Low, X-High.

Job 9 = YOffset

Funktion: Erhöht die Y-Koordinaten von P1 um den angegebenen Wert. Benötigt wird ein Byte (Y).

Job 10 = XYOffset

Funktion: Wie Job 8 und Job 9. Benötigt werden drei Bytes X-Low, X-High, Y.

C139	SetPattern

Zeiger auf gewünschtes Muster setzen. Die Nummer des Musters wird im Akku übergeben (0-33). Über das Aussehen der Muster können Sie sich bei GEOPAINT in der Musterleiste informieren.

C13C	GetScanLine

Berechnet die Startadresse der Zeile, deren Nummer (0-199) im X-Register übergeben wird. Das Ergebnis steht in $0C/$0D für den sichtbaren und in $0E/$0F für den unsichtbaren Bildschirm (vorher in $2F den (die) Bildschirm(e) auswählen!).

C13F	TestPoint

Testet einen Grafikpunkt.

$08,$09	X-Position
$18	Y-Position

Ist der Punkt gesetzt, wird das Carry-Flag gesetzt, andernfalls gelöscht. $2F entscheidet, auf welcher Bildschirmseite getestet wird.

C142	BitmapUp

Zeichnet ein Bild beliebiger Größe. Mit dieser Routine werden beispielsweise die Icons im Desktop gezeichnet. Es müssen folgende Parameter übergeben werden:

$02,$03	Adresse der Daten des Bildes
$04	X-Position (0-39)
$05	Y-Position (0-199)
$06	Breite des Bildes in Bytes
$07	Höhe des Bildes in Zeilen

Die X-Position kann also nur in 8-Bit-Schritten angegeben werden und wird automatisch mit 8 multipliziert. Die Grafikdaten müssen in einer für GEOS spezifischen Art und Weise komprimiert vorliegen. Ein typisches Beispiel hierfür sind die Daten eines Icons im Info-Sektor einer Datei. Hier die notwendigen Informationen zu dem benutzten Format:

Grafikdaten werden in GEOS nicht linear abgelegt, da dies viel zu viel Speicher in Anspruch nehmen würde. Statt dessen wird die Tatsache ausgenutzt, daß in den meisten Grafiken ganze Byte-Folgen gleichartig aufgebaut sind. Diese Byte-Folgen können nun aber zusammengefaßt werden, wodurch man je nach Bildinhalt einige KByte Speicher einsparen kann. Dies funktioniert jedoch nur dann rentabel, wenn man sich von dem durch die Hardware des 64er festgelegten Zusammenhang zwischen Grafikdaten und Speicherstellen löst und die Daten zeilenweise zusammenfaßt. Die Hardware des 64er faßt nämlich immer 8 im Speicher aufeinanderfolgende Bytes zu einem auf dem Bildschirm untereinanderliegenden Grafikblock zusammen. Man kann sich nun leicht vorstellen, daß die Grafikdaten einer waagerechten Grafikzeile eher ein sich wiederholendes Muster aufweisen, als die Daten mehrerer solcher untereinanderliegender Achter-Blocks. Um Daten komprimieren zu können, benötigt man bestimmte Steuer-Bytes, die sich mit den Grafikdaten abwechseln. In GEOS werden Grafikdaten durch ein Steuer-Byte eingeleitet, das folgende Werte annehmen kann:

0 Ende der Grafikdaten.

1-127 Der angegebene Wert entscheidet darüber, wie oft das als nächstes angegebene Byte wiederholt dargestellt werden soll. Im Extremfall kann hiermit also auch wieder eine unkomprimierte Grafik abgelegt werden. Dazu muß sich das Steuer-Byte mit dem Wert "1" immer mit einem Byte der Grafikdaten abwechseln. Dadurch würde sich natürlich der benötigte Speicherplatz genau verdoppeln. Tauchen aber in einer Grafik große, ungemusterte Flächen auf, können mit diesem Steuerzeichen maximal 127 Bytes auf nur 2 Bytes komprimiert werden (also auf nur noch ca. 1.6% des ursprünglichen Speicherplatzbedarfes!).

128-220 Der angegebene Wert, vermindert um 128, gibt an, wie viele Daten nun folgen, die direkt als Grafikdaten übernommen werden sollen. Hierdurch können komplizierte Grafiken, die keine sinnvolle Datenkompression erlauben, abgespeichert werden. Der Speicherplatzverlust durch Steuerzeichen wird hierdurch auf ein akzeptables Maß begrenzt. Schließlich wäre es ja auch möglich, solche Daten mit dem Steuerzeichen "1" zu speichern (s.o.).

221-255 Solche Steuerzeichen sind etwas komplizierter aufgebaut. In GEOS hat man es geschafft, sogar Grafikmuster zu komprimieren, die sich nicht auf ein Byte beschränken. Um dieses Steuerzeichen richtig interpretieren zu können, muß man es erst einmal um 220 vermindern. Der Wert, der dabei übrigbleibt, gibt an, wie viele Bytes das sich wiederholende Muster insgesamt aufweist. Natürlich fehlt noch die Information darüber, wie oft dieses Muster wiederholt werden soll. Aus diesem Grund findet man im nächsten Byte noch keine Grafikdaten, sondern ein weiteres Steuer-Byte. Dieses Steuer-Byte gibt nun an, wie oft das Muster wiederholt werden soll. Erst jetzt folgen die Grafikdaten, die wiederum (Ihnen bleibt aber

auch nichts erspart) in einem der ersten beiden Formate vorliegen müssen. Somit taucht das erste Grafik-Byte erst nach drei Steuerzeichen auf.

C145	PutChar

Gibt den Wert im Akku an der aktuellen Position als ASCII-Zeichen aus. Zeichen kleiner als dez. 32 ($20) werden als Steuerzeichen interpretiert (siehe ASCII-Tabelle im Anhang). Ungültige Werte führen zum Absturz des Rechners.

C148	PutString

PutString gibt eine Zeichenkette mit dem aktuellen Zeichensatz aus. Die Steuerzeichen zur Änderung des Zeichensatzes (siehe ASCII-Tabelle im Anhang) werden ignoriert. Alle anderen Steuerzeichen werden jedoch akzeptiert.

Parameter

$02/$03	Adresse des auszugebenden Textes, der mit Null enden muß.
$05	Y-Position für die Ausgabe
$18/18	X-Position für die Ausgabe

C14B	UseSystemFont

Schaltet auf den residenten BSW-9-Point-Font des Betriebssystems um. Es werden keine Parameter benötigt.

C14E StartMouseMode

Initialisiert die Mausabfrage. Ist das Carry-Flag gesetzt und steht in $18/$19 ein Wert ungleich "0", so werden $18/$19 als neue Mausposition (X) und der Inhalt des Y-Registers als neue Mausposition (Y) übernommen. Die Mausgeschwindigkeit wird auf 0 gesetzt. Unabhängig vom Carry-Flag wird die Menüabfrage initialisiert.

C151 DoMenu

Diese Routine erstellt ein komplettes Pull-Down-Menü. Normalerweise gehört ein Aufruf von DoMenu zu jeder Initialisierungsroutine einer Application in GEOS. Dabei müssen die folgenden Parameter übergeben werden:

$02,$03	Zeiger auf eine Datentabelle

Aufbau der Datentabelle (Format des Menüs):

Byte 0	Y-Position oben
Byte 1	Y-Position unten
Byte 2	X-Position links Low
Byte 3	X-Position links High
Byte 4	X-Position rechts Low
Byte 5	X-Position rechts High
Byte 6	Status-Byte (siehe unten)
Byte 7	Zeiger auf Menübegriff 1 Low
Byte 8	Zeiger auf Menübegriff 1 High
Byte 9	Aufbau-Flag (siehe unten)
Byte 10	Pointer für Menüpunkt 1 Low (siehe unten)
Byte 11	Pointer für Menüpunkt 1 High

Weiter wie ab Byte 7 für alle weiteren Menüpunkte (maximal 15). Das Status-Byte (Byte 6) existiert nur für das Hauptmenü und hat den folgenden Aufbau:

Bit 0-4	Anzahl der Hauptmenüpunkte
Bit 6	1, Die Maus kann die Menügrenzen nicht verlassen. 0, Maus ist frei beweglich.
Bit 7	1, Menüpunkte stehen untereinander, sonst nebeneinander wie im Desktop, GEOPAINT, etc...

Die Bits des Aufbau-Flags haben die folgende Bedeutung:

Bit 7	1, dann zeigt der Pointer dieses Menüpunktes (also z.B. Byte 10 und 11 für den 1. Hauptmenüpunkt) auf die Daten eines Submenü, das wiederum eine vollständige Menüstruktur aufweisen muß. Auf diese Weise können maximal 3 geschachtelte Untermenüs mit je 15 Einträgen erstellt werden.
Bit 7	0 und Bit 6 = 0, der Pointer dieses Menüpunktes zeigt auf die Einsprungadresse der ausgewählten Routine.

Wichtig!

Nach dem Einsprung in die eigene Routine baut sich das Menü nicht selbständig wieder ab. Dem Anwender wird so in seinem Programm die Möglichkeit gegeben, selbst zu entscheiden, was mit dem derzeitigen Menüstatus geschehen soll. Im einfachsten Fall kann man durch den Aufruf der Funktion GotoFirstMenu = $C1BD den Abbau aller Untermenüs veranlassen. Soll jedoch das zuletzt angewählte Untermenü noch weitere Auswahlmöglichkeiten zulassen, kann dies durch den Aufruf der Funktion ReDoMenu = $C193 erreicht werden.

Möchte man jedoch die Auswahl eines weiteren Menüpunktes im derzeit übergeordneten Menü zulassen, kann dies durch den Aufruf der Funktion DoPreviousMenu = $C190 geschehen. In jedem Fall muß eine dieser Routinen vor dem Ende des eigenen Programms durch RTS aufgerufen werden. Nähere Angaben zu diesem Thema finden Sie auch bei der Behandlung der einzelnen Routinen.

Bit 7	0 und Bit 6 = 1, der Pointer des Menüpunktes zeigt auch hier auf eine Einsprungadresse. Nachdem die

ab dort beginnende Routine abgearbeitet worden ist und mit RTS endet, wird ein weiteres Submenü erstellt. Der Zeiger auf die Daten dieses Submenüs muß vor dem RTS in $02,$03 eingetragen werden.

Hierdurch können Jobs ausgeführt werden, während das Menü aktiv ist. Damit ist der Programmierer nicht gezwungen, die komplette Menüstruktur bis zum letzten Submenü schon von vornherein festzulegen. Der Font-Editor benutzt diese Funktion zum Beispiel, um die vorhandenen Point-Höhen für den angewählten Zeichensatz zu laden, bevor er sie im Menü anzeigt. Die einzelnen Menübegriffe müssen alle mit "0" abgeschlossen sein.

C154 RecoverMenu

Sorgt dafür, daß beim Abbau des zuletzt ausgeklappten Menüs der ursprüngliche Bildschirmaufbau wiederhergestellt wird. Im Normalfall braucht sich der Anwender um diese Funktion nicht zu kümmern, da sie Bestandteil der Routinen ist, die ein Menü abbauen (DoPreviousMenu, GotoFirstMenu). Ist der Vektor $84B1/$84B2 gleich Null, dann wird beim Abbau eines Menüs der Hintergrund einfach weiß eingefärbt (nicht sehr sinnvoll). Andernfalls muß in den beiden Speicherstellen die Adresse einer Routine stehen, die den Hintergrund wiederherstellt. Bei der normalen Warmstart-Konfiguration von GEOS steht hier ein Sprung auf die Routine RecoverRectangle (siehe auch dort). Diese holt aus dem unsichtbaren Bildschirm die original Grafikdaten des vom Menü überschriebenen Bereiches und kopiert sie in den sichtbaren Bildschirm.

Da ein Menü nur auf dem sichtbaren Bildschirm ausgegeben wird, kann das Bild auf diese Art restauriert werden. Falls aber nicht genügend Platz für eine vollständige Kopie des Bildschirms ab $6000 vorhanden ist (wie z.B. bei GEOPAINT), kann man den Vektor $84B1/$84B2 auch auf eine eigene Routine umbiegen. Diese muß dafür sorgen, daß die vom Menü überschriebenen Daten vor dem Ausklappen in irgendeiner Form gerettet

und beim Abbau wieder auf den sichtbaren Bildschirm geschrieben werden. Falls möglich, sollte man jedoch mit einem Hintergrundbildschirm arbeiten, da einem in diesem Fall die Programmierung der Rettungsroutinen erspart bleibt.

C157 RecoverAllMenus

Baut nacheinander alle heruntergeklappten Untermenüs ab. Dazu wird RecoverMenu (s.o.) als Unterprogramm benutzt.

Parameter

\$84B1/\$84B2	siehe RecoverMenu

C15A DoIcons

Erstellt anklickbare Bilder wie beispielsweise das Diskettensymbol im DESKTOP. Genauso wie DoMenu wird auch DoIcons in den meisten Fällen Bestandteil einer Programm-Initialisierung sein. Die Icon-Funktion ist ebenso wie die Menüverwaltung in GEOS "event driven". Der Anwender teilt also dem Betriebssystem in einer Tabelle mit, welche Routinen beim Anklicken bestimmter Icons ausgeführt werden sollen. Nach dieser Initialisierung muß das Programm mit einem RTS enden, damit die GEOS-Hauptschleife die Kontrolle übernehmen kann.

Parameter

\$02,\$03	Adresse der Tabelle (Low/High)

Aufbau der Tabelle:

Byte 0	Anzahl der gewünschten Klickfelder
Byte 1/2	Gewünschte X-Mausposition (Low/High)
Byte 3	Gewünschte Y-Mausposition für jedes Klickfeld
Byte 4/5	Zeiger auf das Muster (Low/High)

Byte 6	X-Position des linken Randes des Icons (0-39)
Byte 7	Y-Position des oberen Randes des Icons (0-199)
Byte 8	Breite des Icons in Bytes
Byte 9	Höhe des Icons in Zeilen
Byte 10/11	Job-Adresse bei Klick (Low/High)

Ab hier wie Byte 4-11 für alle weiteren Icons.

Nach dem Zeichnen der Bilder auf dem Bildschirm werden die Klickfelder aktiviert und ständig auf Anklicken getestet.

Sollte ein bestimmtes Icon angeklickt worden sein, so ruft die GEOS-Hauptschleife das in den entsprechenden Stellen eingetragene Unterprogramm auf (für das erste Icon muß diese Adresse also in Byte 10 und 11 der Tabelle stehen).

Hier wird nun der gewünschte Vorgang ausgeführt und anschließend mit RTS wieder zur Hauptschleife zurückgekehrt.

Innerhalb der Routine können aus den Speicherstellen $02 und $03 die folgenden Informationen ausgelesen werden:

$02 Nummer des angeklickten Icons (0 bis N). Dies ist sehr nützlich, falls alle Icons bis auf unterschiedliche Parameter die gleiche Funktion ausführen sollen.

In einem solchen Fall kann man nämlich für alle Icons die gleiche Job-Adresse eintragen und hier durch Auswertung der Speicherstelle $02 Icon-spezifische Korrekturen der Parameter vornehmen.

$03 $FF, dann ist das Icon zweimal kurz hintereinander angeklickt worden (Doppelklick).

$03 $00, dann ist das Icon nur einmal oder mit zu großem zeitlichen Abstand zweimal angeklickt worden.

Ferner kann der Anwender durch das Eintragen bestimmter Werte in die Speicherstelle $84B5 entscheiden, ob, und wenn ja, wie das Anklicken eines Icons dem Benutzer mitgeteilt werden soll:

Bit 6 = 0 und Bit 7 = 0:
In diesem Fall wird einfach nur die angegebene Routine angesprungen. Am Aussehen des Icons ändert sich also beim Anklicken nichts.

Bit 6 = 0/1 und Bit 7 = 1:
Dann wird das Icon für einen kurzen Augenblick invertiert und anschließend zurückinvertiert.
Dieses Verfahren findet zum Beispiel bei den OK- oder CANCEL-Icons Verwendung.

Bit 6 = 1 und Bit 7 = 0:
Bei dieser Einstellung wird das angeklickte Icon invertiert. Beispiel: File-Icons im DESKTOP.

C15D	DShiftLeft

Das X-Register muß auf eine 2-Byte-Adresse auf der Zeropage zeigen, deren Inhalt so oft um ein Byte nach links geschoben wird, wie es der Inhalt des Y-Registers angibt (jedesmal * 2).

C160	BBMult

Multipliziert 2 einzelne Bytes auf der Zeropage. Als Zeiger auf diese Bytes dienen das X- und das Y-Register.

Das Low-Byte des Ergebnisses steht anschließend in der durch das X-Register bezeichneten Speicherstelle, das High-Byte befindet sich direkt dahinter.

C163	BMult

Multipliziert 2-Byte-Wert (Wort) mit einem Byte.

Parameter

X-Register Zeigt auf Zeropage-Adresse, die das Wort enthält.
Y-Register Zeigt auf Zeropage-Adresse, die das Byte enthält.

Das 16-Bit-Ergebnis steht anschließend in den durch das X-Register bezeichneten Speicherstellen (Low/High).

C166	DMult

Multipliziert einen 2-Byte-Wert (Wort) mit 2-Byte-Wert (Wort).

Parameter

X-Register Zeigt auf Zeropage-Adresse, die Wort 1 enthält
Y-Register Zeigt auf Zeropage-Adresse, die Wort 2 enthält.

Das 16-Bit-Ergebnis steht anschließend in den durch das X-Register bezeichneten Speicherstellen (Low/High).

C169	DDiv

Dividiert einen 2-Byte-Wert (Wort) durch 2-Byte-Wert (Wort).

Parameter

X-Register Zeigt auf Zeropage-Adresse, die Wort 1 enthält.
Y-Register Zeigt auf Zeropage-Adresse, die Wort 2 enthält.

Das 16-Bit-Ergebnis steht anschließend in den durch das X-Register bezeichneten Speicherstellen (Low/High).

Falls bei der Division ein Rest auftritt, wird dieser in $12/13 (Low/High) abgelegt. Damit ist in GEOS auch eine Modulo-Funktion verfügbar.

C16C	DSdiv

Diese Funktion unterscheidet sich von DDiv nur darin, daß die beiden Argumente als vorzeichenbehaftet (signed) angesehen werden.

Damit kann ein 16-Bit-Wort Werte zwischen -32768 und +32767 annehmen, während ein "normales" 16-Bit-Wort (unsigned) zwischen 0 und 65535 liegt (jeweils einschließlich).

Parameter

Siehe DDiv.

C16F	Dabs

Berechnet aus einer vorzeichenbehafteten 16-Bit-Zahl ihren Absolutwert.

Parameter

X-Register Zeigt auf die Zeropage-Adresse, die das vorzeichenbehaftete Wort enthält.

Das 16-Bit-Ergebnis steht anschließend in den durch das X-Register bezeichneten Speicherstellen (Low/High).

C172 Dnegate

Bewirkt eine Vorzeichenumkehr bei einem vorzeichenbehafteten 16-Bit-Wort.

X-Register Zeigt auf die Zeropage-Adresse, die das vorzeichenbehaftete Wort enthält.

Das 16-Bit-Ergebnis steht anschließend in den durch das X-Register bezeichneten Speicherstellen (Low/High).

C175 Ddec

Vermindert ein nicht vorzeichenbehaftetes 16-Bit-Wort um eins.

X-Register Zeigt auf die Zeropage-Adresse, die das vorzeichenbehaftete Wort enthält.

Das 16-Bit-Ergebnis steht anschließend in den durch das X-Register bezeichneten Speicherstellen (Low/High).

C178 ClearRam

Füllt einen Speicherbereich mit "0".

Parameter

$02,$03 Anzahl Bytes, die gelöscht werden sollen
$04,$05 Startadresse des zu löschenden Bereiches

C17B	FillRam

Füllt einen Speicherbereich mit dem Wert aus $06.

Parameter

$02,$03	Anzahl Bytes
$04,$05	Startadresse
$06	Schreibwert

C17E	MoveData

Transportiert einen Speicherbereich an die angegebene Adresse.

Parameter

$02,$03	Ursprungsadresse
$04,$05	Zieladresse
$06,$07	Anzahl Bytes

C181	InitRam

Kopiert Speicherblöcke, die in einer Tabelle näher bezeichnet werden müssen. In $02,$03 muß der Zeiger auf diese Tabelle übergeben werden. Diese Routine ist sehr nützlich, um in einer Initialisierungsroutine bestimmte Speicherstellen mit Startwerten vorzubelegen.

Aufbau der Tabelle

Byte 0/1	Zieladresse (Low/High)
Byte 2	Anzahl der Bytes in diesem Block
Byte 3	ab hier steht der Datenblock

Hinter dem Datenblock geht es weiter wie bei Byte 0 für den nächsten Block. Das Ende der Tabelle muß durch die Zieladresse $0000 gekennzeichnet werden.

C184	PutDecimal

Mit Hilfe dieser Routine kann eine 16-Bit-Dezimalzahl ausgegeben werden (vorher in $2F den Bildschirm auswählen!).

Parameter

$02/$03	16-Bit-Zahl, die ausgegeben werden soll (Low/High)
$05	Y-Position, an der die Ausgabe stattfinden soll
$18/$19	Dito für die X-Position (Low/High)
Akku	Bit 7 = 1: Dann erfolgt die Ausgabe linksbündig, andernfalls rechtsbündig. Bit 6 = 1: Um führende Nullen zu unterdrücken Bits 0-5: Müssen Angaben über die Breite des gewünschten Ausgabefeldes (0 bis 63) enthalten, falls die Ausgabe rechtsbündig erfolgen soll (Bit 7 = 0).

C187	GetRandom

Erzeugt in den Speicherstellen $850A/$850B eine Art Zufallszahl, die zwischen 0 und 65220 (incl.) liegen kann. Der Wert wird durch einen einfachen Algorithmus erzeugt und kann daher nicht als echte Zufallszahl angesehen werden.

C18A	MouseUp

Schaltet die Maus nach einem MouseOff wieder ein. Sämtliche Mausparameter entsprechen dem Zustand vor dem Aufruf von MouseOff.

C18D MouseOff

Schaltet die Mausfunktion aus und löscht den Mauspfeil.

C190 DoPreviousMenu

Baut das zuletzt ausgeklappte Untermenü wieder ab und erlaubt eine erneute Auswahlmöglichkeit aus dem übergeordneten Menü. Bevor diese Routine aufgerufen wird, muß die Mausposition auf ein Feld innerhalb des übergeordneten Menüs gesetzt werden. Andernfalls befindet sich die Maus ja außerhalb der Menüfelder, was nach einem Aufruf von DoPreviousMenu zu einem sofortigen Abbau sämtlicher Submenüs führen würde.

Parameter

\$3A/\$3B	Aktuelle X-Position der Maus (Low/High)
\$3C	Aktuelle Y-Position der Maus

C193 ReDoMenu

Baut das aktuelle Untermenü wieder auf. Dadurch ist es möglich, aus einem Untermenü mehrere Punkte auszuwählen.

C196 GetSerialNumber

Holt die 16-Bit-Seriennummer des GEOS-Kernals in die Speicherstellen \$02/\$03. Diese Zahl wird nur ein einziges Mal (und zwar beim ersten Booten von GEOS) erzeugt und fest in das Kernal-File auf der Diskette eingetragen. In der GEOS-Version 1.3 findet man diese Nummer in den Speicherstellen \$9EA7 und

$9EA8. Nach der ersten Installation wird außerdem die Speicherstelle $9EA9 auf $FF gesetzt, damit der Installationsteil bei späteren Boot-Vorgängen übersprungen wird.

Sämtliche käuflichen Zusatzprogramme (z.B. GEODEX) fragen bei ihrer ersten Installation mehr oder weniger direkt über die Routine GetSerialNumber die GEOS-Seriennummer ab und speichern sie fest auf der Diskette ab. Damit ist es nun so gut wie ausgeschlossen, daß dieses Zusatzprogramm in Verbindung mit einer anderen GEOS-Systemdiskette funktioniert. Die Wahrscheinlichkeit, auf zwei gleiche Seriennummern zu treffen, beträgt nämlich genau 1 zu 65535 (Die Seriennummer $0000 gibt es nicht!). Da jedes Programm zusätzlich auch noch mit einem Kopierschutz versehen ist, genügt es auch nicht, eine Kopie der Diskette vor der ersten Installation anzufertigen.

C199 Sleep

Bewirkt eine Verzögerung im Programmablauf. Die Verzögerungszeit (in 1/50 Sekunden) wird in $02,$03 übergeben und die Routine durch JSR Sleep aufgerufen. Nach Ablauf der Wartezeit wird das Programm automatisch hinter dem JSR fortgesetzt. Der Vorteil gegenüber einer einfachen Warteschleife besteht darin, daß die Wartezeit anderen Jobs der Hauptschleife zur Verfügung gestellt wird. Damit sieht ein typischer Programmausschnitt für eine Verzögerung in GEOS folgendermaßen aus: (Verzögerung: 20 Sekunden = 1000/50 Sekunden -> $03E8)

```
...
...
LDX #$E8
LDY #$03
STX $02
STY $03
JSR $C199
... (Fortsetzung des Programms nach 20 Sekunden)
...
```

Man sollte jedoch beachten, daß selbstverständlich in der Zwischenzeit von anderen Routinen, die von der Hauptschleife aufgerufen werden, Manipulationen an Speicherinhalten vorgenommen werden können.

C19C — ClearMouseMode

Setzt das Maus-Flag ($30) auf "0" und schaltet das Sprite ab. Dadurch reduziert sich die Zeit, in der der Video-Chip zur Darstellung des Sprites auf das RAM zugreifen muß. Die Turbo-Disk-Routinen schalten bei Lade- und Speichervorgängen kurzzeitig die Maus ab, da die Datenübertragung durch den dauernden Zugriff des Video-Chips auf das RAM gestört würde. Normalerweise müßte hierbei sogar der gesamte Video-Chip ausgestellt werden, da zur Darstellung der Hires-Grafik auch Speicherzugriffe nötig sind. Die Turbo-Disk-Routinen nutzen jedoch geschickt die Zeitintervalle aus, in denen mit Sicherheit keine solchen Zugriffe zu erwarten sind. Das Gegenstück zu ClearMouseMode ist StartMouseMode ($C14E).

C19F — iRectangle

Im Gegensatz zur Funktion "Rectangle" können hier die Daten direkt hinter den Befehl "JSR iRectangle" gelegt werden. Das sieht dann folgendermaßen aus:

```
1000 JSR $C19F
1003 Byte 1 für $06 (oben)
1004 Byte 2 für $07 (unten)
1005 Byte 3 für $08 (links)
1006 Byte 4 für $09    "
1007 Byte 5 für $0A (rechts)
1008 Byte 6 für $0B    "
1009 hier wird das Programm fortgesetzt.
```

Für das Füllmuster gilt das unter "Rectangle" ($C124) Gesagte. Die Fortsetzung des Programms erfolgt anschließend automatisch direkt hinter den Daten.

C1A2	**iFrameRectangle**

Funktion wie FrameRectangle ($C127). Die Daten müssen jedoch direkt hinter dem Unterprogramm-Aufruf stehen.

C1A5	**iRecoverRectangle**

Funktion wie RecoverRectangle ($C12D). Die Daten müssen jedoch direkt hinter dem Unterprogramm-Aufruf stehen.

C1A8	**iGraphicsString**

Funktion wie GraphicsString ($C136). Jobs und Daten müssen jedoch direkt hinter dem Unterprogramm-Aufruf stehen.

C1AB	**iBitmapUp**

Funktion wie BitmapUp ($C142). Die Daten müssen jedoch direkt hinter dem Unterprogramm-Aufruf stehen.

Baut die gesamte Menüstruktur bis zum Hauptmenü ab.

Normalerweise sollte diese Routine zu Beginn jedes Programms stehen, das vom Menü aufgerufen wird (siehe auch DoMenu = $C151).

Schaltet einen neuen Zeichensatz ein. Der Zeichensatz muß sich schon im Speicher befinden.

Die Startadresse wird in $02,$03 übergeben.

C1AE	iPutString

Funktion siehe $C148; die Daten hinter dem Unterprogramm-Aufruf haben das folgende Format:

Byte 0/1 X-Position (Low/High)
Byte 2 Y Position
Byte 3 und folgende: Text für die Ausgabe. Der Text muß mit einer "0" abgeschlossen werden.

C1C0	InitTextPrompt

Legt die Länge des Cursor-Strichs abhängig vom geladenen Zeichensatz fest. Dazu muß im Akku die Höhe des Zeichensatzes

C1CF	PosSprite

Das Sprite wird an die vorgegebene Stelle positioniert.

$08 Nummer der Sprites
$0A/$0B X-Position (0-319) (Low/High)
$0C Y-Position (0-199)

C1D2	EnableSprite

Schaltet das Sprite, dessen Nummer in $08 steht, ein.

C1D5	DisableSprite

Schaltet das Sprite, dessen Nummer in $08 steht, aus.

C1D8	CallRoutine

Ersetzt im Befehlssatz des 6502er fehlenden Befehl JSR ($xxxx). CallRoutine ermöglicht indirekten Einsprung in ein Unterprogramm. Die Anfangsadresse muß im Akku (Low) und X-Register (High) übergeben und JSR CallRoutine aufgerufen werden.

C1DB	CalcBlocksFree

Berechnet die Anzahl der freien Blöcke aus der BAM. Der Zeiger auf die BAM muß sich in $0C,$0D befinden (normalerweise liegt die BAM in GEOS ab $8200). Ergebnis in $0A,$0B.

C1DE ChkDkGEOS

Testet die BAM auf GEOS-Format. Das Ergebnis wird im Akku und in $848B übergeben. Ist der Akku = 0, hat die Diskette kein GEOS-Format, bei $FF handelt es sich um eine GEOS-Diskette. Die aktuelle BAM befindet sich in GEOS normalerweise ab $8200. Die Routine ChkDkGEOS erwartet in $0C/$0D einen Zeiger auf die BAM.

C1E1 NewDisk

Lädt die BAM einer neu eingelegten Diskette nach $8200.

Parameter

Keine

Rückgabe: Error-Status im X-Register (siehe Anhang)

C1E4 GetBlock

Liest einen vollständigen Sektor (256 Bytes) von der Diskette.

Parameter

$04	Track (1-35)
$05	Sektor (0-Maximal-Sektor, siehe Floppy-Bedienungsanleitung)
$0A,$0B	Zeiger auf einen freien Datenpuffer (meist $8000)

Rückgabe: Error-Status im X-Register (siehe Anhang)

C20B	FindFile

Sucht im Directory und im Border-Block (das ist der Sektor, in dem GEOS die File-Einträge verwaltet, deren Icons auf dem Rand abgelegt sind) nach einem File-Eintrag. Dieser wird dann in $8400 übergeben.

$0E,$0F	Zeiger auf den File-Namen (max. 16 Stellen plus eine Abschluß-Null)

Im X-Register wird die Fehlermeldung übergeben. (X = 0: kein Fehler aufgetreten, X = 5: File not Found, weitere Fehler: siehe Anhang).

C20E	CRC

Berechnet eine Prüfsumme über einen definierten Bereich des Arbeitsspeichers. Diese Funktion ist zum Beispiel recht nützlich, um Kopierschutz-Routinen vor Manipulationen zu sichern.

Parameter

$02/$03	Zeiger auf die Anfangsadresse der Daten
$04/$05	Umfang der Daten in Bytes

Das Ergebnis steht anschließend in $06/$07.

C211	LdFile

Dies ist eine Laderoutine für GEOS-Files. Ihr muß in $14,$15 ein Zeiger auf den kompletten File-Eintrag übergeben werden (beispielsweise durch FindFile $C20B nach $8400 holen).

LdFile holt die notwendigen Informationen aus dem INFO-Sektor, lädt das File an die dort eingetragene Stelle und startet gegebenenfalls das Programm (je nach File-Typ). Eine alternative Ladeadresse kann in dieser Low-Level-Ebene nicht mehr angegeben werden (siehe GetFile).

C214	EnterTurbo

Schaltet die schnellen Disk-Routinen ein. Falls sich die Routinen noch nicht im Floppy-RAM befinden, werden sie erst dorthin kopiert.

Parameter

$8489 Nummer des Laufwerks, auf das sich EnterTurbo bezieht (8 oder 9). Bei nur einem angeschlossenen Laufwerk steht hier automatisch eine Acht.

C217	LdDeskAcc

Kompletter Ladevorgang für ein Accessory. LdDeskAcc wird beispielsweise automatisch aufgerufen, wenn man den File-Namen eines Accessories an GetFile = $C208 übergibt. Diese Routine erstellt ein Swap-File, rettet den derzeitigen Systemstatus (Menus, Prozesse, alle Vektoren von $849B bis einschl. $84C1, aktueller Font, Style, Füllmuster), lädt und startet das Accessory. Trägt das Accessory den Job RstrAppl = $C23E in den Main-Vector $849B/$849C ein und führt es anschließend einen RTS aus, so wird der ursprüngliche Zustand der Application wiederhergestellt (siehe auch GetFile = $C208).

C24D	NxtBlkAlloc

Diese Routine unterscheidet sich von BlkAlloc = $C1FC nur durch eine etwas höhere Einsprungadresse. Dadurch besteht die Möglichkeit, in $08,$09 den Track und Sektor selbst festzulegen, ab dem die Suche nach freien Blöcken beginnen soll. Für normale Anwendungen ist dieser Befehl jedoch nicht erforderlich.

C250	ImprintRectangle

Routine rettet ein Rechteck auf den "unsichtbaren" zweiten Bildschirm (Gegenstück: RecoverRectangle= $C12D).

$06	Obere Grenze
$07	Untere Grenze
$08,$09	Linke Grenze (Low,High)
$0A,$0B	Rechte Grenze (Low,High)

C253	iImprintRectangle

Wie ImprintRectangle = $C250. Die acht Daten müssen jedoch direkt hinter dem Unterprogramm-Aufruf liegen.

C256	DoDlgBox

Erzeugt ein Fenster (Dialog Box) für Ein- und/oder Ausgaben. Der Routine muß hierzu nur in $02/$03 ein Pointer auf eine Datentabelle übergeben werden, die nähere Informationen über das Fenster enthält. Solch eine Tabelle ist folgendermaßen aufgebaut:

1. Byte	Dieses Byte enthält im 7. Bit Informationen über die Größe der Dialogbox: Bit 7 = 1: Fenstergröße und Position wie bei den üblichen DESK-TOP-Fenstern. In den meisten Fällen wird man diesen Modus wählen. Bit 7 = 0: Die sechs folgenden Bytes der Tabelle enthalten Angaben über die Größe und Position der Dialogbox. Bits 0-4: Hier kann die Nummer des Musters angegeben werden, mit dem die Dialogbox hinterlegt werden soll (Schattenfenster). Der Wert "0" ergibt kein Schattenfenster. DESK-TOP-Dialogboxen tragen hier eine Eins ein (schwarzer Schatten).
2. Byte	Position des oberen Fensterrandes (0-199)(nur, falls im 1. Byte Bit 7 = 0 ist)
3. Byte	Position des unteren Fensterrandes (0-199, > 2. Byte)(nur, falls im 1.Byte Bit 7 = 0 ist)
4./5. Byte	Position des linken Fensterrandes (0-319)(nur, falls im 1. Byte Bit 7 = 0 ist)
6./7. Byte	Position des rechten Fensterrandes (0-319, > 4. und 5. Byte)(nur, falls im 1.Byte Bit 7 = 0 ist)

Die weiteren Bytes der Fenstertabelle können aus den nun folgenden Steuerzeichen bestehen, die in beliebiger Reihenfolge auftreten dürfen und vom Betriebssystem nacheinander abgearbeitet werden:

$00

Funktion:	Endekennzeichen der Tabelle.
Parameter:	keine
Kommentar:	Mit diesem Zeichen muß jede Tabelle enden. Die Reaktion des Rechners hängt von der Art der vorausgegangenen Steuerzeichen ab. Durch das Endekennzeichen wird kein Abbau der Dialogbox veranlaßt. Hierfür existieren eigene Steuerzeichen.

$01

Funktion:	Das OK-Feld soll ausgegeben werden.
Parameter:	1. Byte: Abstand vom linken Rand der Dialogbox in ganzen Bytes 2. Byte: Abstand vom oberen Rand der Dialogbox in Zeilen
Kommentar:	Es erscheint an der angegebenen Position das auch vom DESKTOP benutze OK-Icon. Das Fenster wird automatisch abgebaut, sobald der Anwender das Feld mit der Maus anklickt. DoDlgBox kehrt anschließend zum aufrufenden Programm zurück. Hier kann nun aus der Speicherstelle $02 die Nummer des Steuerzeichens ausgelesen werden, die den Abbau des Fensters verursacht hat (in diesem Fall also eine $01).

$02

Funktion:	Das CANCEL-Feld (Abbruch) soll ausgegeben werden.
Parameter:	Wie bei Steuerzeichen $01.
Kommentar:	Wie bei Steuerzeichen $01.

$03

Funktion:	Das YES-Feld soll ausgegeben werden.
Parameter:	Wie bei Steuerzeichen $01.
Kommentar:	Wie bei Steuerzeichen $01.

$04

Funktion:	Das NO-Feld soll ausgegeben werden.
Parameter:	Wie bei Steuerzeichen $01.
Kommentar:	Wie bei Steuerzeichen $01.

$05

Funktion:	Das OPEN-Feld (Öffnen) soll ausgegeben werden.
Parameter:	Wie bei Steuerzeichen $01.
Kommentar:	Wie bei Steuerzeichen $01.

$06

Funktion:	Das DISK-Feld soll ausgegeben werden.
Parameter:	Wie bei Steuerzeichen $01.

Kommentar: Wie bei Steuerzeichen $01.

$07-$0A

Diese Steuerzeichen sind in GEOS noch nicht implementiert.

$0B

Funktion: Gibt einen vorhandenen Text in dem Fenster aus.

Parameter: 1. Byte: Abstand des Textes vom linken Rand der Dialogbox in Pixeln
2. Byte: Abstand des Textes vom oberen Rand der Dialogbox in Zeilen
3./4. Byte: Adresse des Textes, der ausgegeben werden soll.

Kommentar: Der Text darf alle darstellbaren ASCII-Zeichen und Style-Kommandos enthalten (siehe ASCII-Tabelle im Anhang). Als Endekennzeichen dient eine Null.

$0C

Funktion: Gibt einen vorhandenen Text aus, auf den über eine Zeropage-Adresse zugegriffen wird.

Parameter: 1. Byte: Abstand des Textes vom linken Rand der Dialogbox in Pixeln
2. Byte: Abstand des Textes vom oberen Rand der Dialogbox in Zeilen
3. Byte: Angabe einer Zeropage-Adresse, die einen Zeiger auf den auszugebenden Text enthalten muß.

Kommentar: Der Text darf alle darstellbaren ASCII-Zeichen und Style-Kommandos enthalten (siehe ASCII-Tabelle im Anhang). Als Endekennzeichen dient eine Null. Erlaubter Bereich für den Pointer in der Zeropage: $0C-$16 (incl.)

$0D

Funktion: Gibt einen vorhandenen Text aus, auf den über eine Zeropage-Adresse zugegriffen wird, und erlaubt eine Korrektur des Textes mittels Tastatur.

Parameter: 1. Byte: Abstand des Textes vom linken Rand der Dialogbox in Pixeln

	2. Byte: Abstand des Textes vom oberen Rand der Dialogbox in Zeilen
	3. Byte: Angabe einer Zeropage-Adresse, die einen Zeiger auf den auszugebenden Text enthalten muß.
	4. Byte: Maximale Anzahl Zeichen, die bei der Eingabe akzeptiert werden sollen
Kommentar:	Der auszugebende Text sollte sinnvollerweise nur aus darstellbaren ASCII-Zeichen bestehen (siehe ASCII-Tabelle im Anhang). Als Endekennzeichen dient eine Null. Erlaubter Bereich für den Pointer in der Zeropage: $0C-$16 (incl.)

$0E

Funktion:	Das Fenster wird abgebaut, sobald die Maus an einer beliebigen Position geklickt wird, an der sich kein anderes Klick-Feld befindet.
Parameter:	keine

$0F

Funktion:	Gibt einen Grafik-String aus.
Parameter:	1. und 2. Byte: Zeiger auf eine Tabelle, die Angaben über den Grafik-String enthalten muß. Für den Aufbau einer solchen Tabelle siehe GraphicsString = $C136.

$10

Funktion:	Ausgabe eines Verzeichnisses, das die Dateinamen aller Files eines bestimmten File-Typs der aktuellen Diskette enthält. Eine solche Datei kann mit der Maus selektiert werden.
Parameter:	1. Byte: Abstand des Verzeichnisses vom linken Rand der Dialogbox in ganzen Bytes
	2. Byte: Abstand des Verzeichnisses vom oberen Rand der Dialogbox in Zeilen
	In $10: Nummer des gesuchten GEOS-File-Typs
	In $16/$17: Pointer auf den CLASS-Namen der gesuchten Dateien. Es werden nur die Dateinamen im Verzeichnis ausgegeben, deren Files im Info-Sektor die spezifizierte Eintragung besitzen. Sollte

dies nicht erwünscht sein, muß der Vektor $16/$17 auf Null gesetzt werden.
In $0C/$0D: Pointer auf einen Puffer, der genügend Platz für den selektierten File-Eintrag (17 Bytes) aufweisen muß. In diesen Puffer überträgt DoDlgBox den selektierten File-Namen.

Kommentar: Zusätzlich zu dem Steuerzeichen $10 sollten in der Tabelle der Dialogbox noch weitere Steuerzeichen auftreten, die ein Verlassen des Fensters ermöglichen (z.B. OK, CANCEL).

$11

Funktion: Ermöglicht den Aufruf eines Unterprogramms des Anwenders, sobald die Maus an einer beliebigen Stelle außerhalb eines Fenster-Icons angeklickt wird.

Parameter: 1.und 2. Byte: Adresse der Routine des Anwenders, die angesprungen werden soll (Low/High).

Kommentar: Nach der Ausführung der angegebenen Routine kann die Dialogbox durch den Aufruf JMP RstrFrmDialog = $C2BF wieder abgebaut werden. Das Programm wird anschließend hinter JSR DoDlgBox fortgesetzt. Wird die angegebene Routine jedoch durch ein RTS beendet, erfolgt eine weitere Abarbeitung der in der Fenstertabelle enthaltenen Steuerzeichen.

$12

Funktion: Ermöglicht die Einbindung selbst definierter Icons in das Fenster.

Parameter: 1. Byte: Abstand des Icons vom linken Rand der Dialogbox in ganzen Bytes
2. Byte: Abstand des Icons vom oberen Rand der Dialogbox in Zeilen
3./4. Byte: Pointer zu den Grafikdaten des selbst definierten Icons (Format: s. BitmapUp = $C142).
5./6. Byte: Null
7. Byte: Breite des Icons in Bytes (Standard: $06)
8. Byte: Höhe des Icons in Zeilen (Standart: $12)

	9./10. Byte: Pointer auf eine Routine, die bei Anklicken des Icons angesprungen werden soll.
$13	
Funktion:	Ermöglicht die Einbindung einer eigenen Routine, die in jedem Fall beim Aufbau des Fensters aufgerufen wird.
Parameter:	1. und 2. Byte: Pointer auf eine Routine, die in jedem Fall beim Aufruf von DoDlgBox abgearbeitet werden soll.
Kommentar:	Der Zeitpunkt der Abarbeitung wird direkt durch die Abfolge der Steuerzeichen innerhalb der Fenstertabelle bestimmt. Folgt also zuerst das Steuerzeichen $01 und anschließend das Steuerzeichen $13, so wird erst das OK-Icon gezeichnet, bevor die angegebene Routine aufgerufen wird. Erst wenn diese mit einem RTS endet, werden alle weiteren Steuerzeichen abgearbeitet.

C259 RenameFile

Ermöglicht das Umbenennen einer Datei auf Diskette.

Parameter

$02,$03	Zeiger auf den derzeitigen File-Namen
$0C,$0D	Zeiger auf den neuen File-Namen

Beide File-Namen müssen jeweils mit einer Null enden.

C25C InitForIO

Wird vor jeder Aktion mit dem schnellen Datenbus aufgerufen und rettet wichtige Parameter (IRQ, NMI etc.). Für Program-

mierer ist der Befehl nicht sehr interessant, da er automatisch von übergeordneten Routinen aufgerufen wird.

C25F	DoneWithIO

Gegenstück zu InitTurboBus ($C25C). Wird von übergeordneten Routinen, die auch InitForIO aufrufen, selbständig ausgeführt.

C262	DShiftRight

Das X-Register muß auf eine 2-Byte-Adresse auf der Zeropage zeigen, die Y-mal nach rechts geschoben wird (gibt jedesmal / 2). Gehört zur Mathe-Bibliothek (ab $C15D).

C265	CopyFString

Kopiert einen String (abgeschlossen durch eine Null) in einen beliebigen Speicherbereich.

X-Register	Zeigt auf Pointer in der Zeropage, der auf den String zeigen muß.
Y-Register	Zeigt auf Pointer in der Zeropage, der auf die Zieladresse zeigt.

Maximal können 256 Bytes kopiert werden.

C268	CopyString

Funktion und Parameter wie CopyFString. Es können jedoch mehrere Texte, die hintereinander liegen, gleichzeitig kopiert

werden. Dazu muß im Akku die Anzahl der Texte übergeben werden. (Jeder Text muß mit Null abschließen!)

C26B	CmpFString

Vergleicht zwei Strings miteinander. Parameter wie CopyFString. Falls im Akku anschließend $00 steht, sind beide Strings gleich. Ansonsten steht hier der Offset auf das ungleiche Element.

C26E	CmpString

Wie CmpFString. Es können jedoch mehrere Texte, die direkt hintereinander liegen, verglichen werden. Dazu muß der Routine im Akku noch die Anzahl der Texte übergeben werden.

Ergebnis wie bei CmpFString. Das X-Register enthält die Nummer des Strings, bei der Ungleichheiten aufgetreten sind.

C271	FirstInit

Führt einen Warmstart von GEOS durch und startet eine Anwendung, die ab der Adresse in $10/$11 liegen muß.

C274	OpenRecordFile

Eröffnet eine schon existierende VLIR-Datei, die sich auf der Diskette befinden muß.

Parameter

$02,$03 Zeiger auf den File-Namen der VLIR-Datei

Rückgabe: Error-Status im X-Register (siehe Anhang)

C277	CloseRecordFile

Schließt eine VLIR-Datei. Dabei wird der Dir-Eintrag aktualisiert (Datum, Uhrzeit). Keine Parameter notwendig.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C27A	NextRecord

Positioniert auf den nächsten Record.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C27D	PreviousRecord

Positioniert auf den vorherigen Record.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C280	PointRecord

Positioniert auf Record, Akku muß dessen Nummer enthalten.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C283	DeleteRecord

Entfernt den aktuellen Record aus der Liste. Die Liste wird dazu aufgerückt, und die durch den Eintrag belegten Blöcke werden wieder freigegeben. Es sind keine direkten Parameter zu übergeben.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C286	InsertRecord

Fügt einen neuen Record vor den aktuellen Record ein. Dieser neue ist anschließend der aktuelle Record.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C289	AppendRecord

Fügt einen neuen Record hinter den aktuellen Record ein. Dieser ist anschließend der aktuelle Record.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C28C	ReadRecord

Liest den aktuellen Record ein. In $10,$11 muß die gewünschte Ladeadresse stehen. In $06,$07 muß die maximale Anzahl gewünschter Daten enthalten sein.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C28F	WriteRecord

Speichert den aktuellen Record ab.

$06,$07 Anzahl Bytes
$10,$11 Startadresse der Daten

Rückgabe: Error-Status im X-Register (siehe Anhang)

C292	SetNextFree

Sucht den nächsten freien Block auf der Diskette und markiert ihn in der BAM als belegt.

Parameter

$08,$09 Zeiger auf den ersten Block (Track und Sektor), ab dem gesucht werden soll.

Der Track 18 (Directory!) wird automatisch übersprungen, falls nicht ausdrücklich in $08 eine 18 eingetragen wird.

C295	UpdateRecordFile

Die folgenden Daten eines geöffneten VLIR-Files werden auf der Diskette aktualisiert:

1. VLIR-Linker
2. Directory-Eintrag (Datum, Zeit)
3. BAM

C298	GetPtrCurDkNm

Setzt einen Zeiger auf den Namen der aktuellen Diskette. Das X-Register muß dazu auf einen Pointer in der Zeropage zeigen, der anschließend auf den Namen zeigt.

C29B	PromptOn

Schaltet den Cursor an der aktuellen Schreibposition ein.

C29E	PromptOff

Schaltet den Cursor aus.

C2A1	OpenDisk

Initialisiert eine neue Diskette. Diese Routine muß nach jedem Diskettenwechsel aufgerufen werden, da das Betriebssystem einen Diskettenwechsel nicht selbst erkennt.

C2A4	DoInlineReturn

Unterprogramm aller Inline-Routinen (das sind die Routinen der Sprungtabelle, die mit einem "i" beginnen, z.B. iRectangle).

C2A7	GetNextChar

Holt ein Zeichen aus dem Tastaturpuffer in den Akku. Ist der Puffer leer, wird eine Null übergeben. Diese Routine ist als Ergänzung zu dem Event-Vektor $84A3 zu verstehen, dessen Adresse jedesmal angesprungen wird, sobald eine Taste betätigt wird. Die dabei aufgerufene Routine kann sich durch GetNextChar weitere Zeichen aus dem Tastaturpuffer holen, bis ihr im Akku eine Null übergeben wird.

C2AA	BitmapClip

BitmapClip ermöglicht es, über eine im nicht sichtbaren Bereich liegende Grafik eine Maske mit einem rechteckigen Ausschnitt zu legen und den Teil der Grafik, der durch die Maske nicht verdeckt wird, auf dem sichtbaren Bildschirm darzustellen. Hierdurch können Grafiken beliebiger Größe ausschnittsweise auf einem Teil des Bildschirms dargestellt werden.

Parameter:

$02,$03	Zeiger auf den Start der Grafikdaten
$04	Linker Rand des Ausgabefensters (in Bytes)
$05	Oberer Rand des Ausgabefensters (in Zeilen)
$06	Breite des Ausgabefensters in Bytes
$07	Höhe des Ausgabefensters in Zeilen

Mit diesen Angaben ist die Größe der ausmaskierten Grafik definiert. Es fehlen jetzt jedoch noch die Angaben, mit denen man den Bereich der Grafik markieren kann, der im Ausgabefenster erscheinen soll:

$18	Angabe, wie viele Bytes (= je 8 Pixel) auf der linken Seite der Gesamtgrafik übersprungen werden sollen, bis die Ausgabe im Fenster beginnt. Der rechte Rand ergibt sich automatisch aus den Angaben in $06.

$19	Hier muß die Anzahl Bytes der Grafik angegeben werden, die rechts neben dem Ausgabefenster ausmaskiert wird. Durch Addition der Werte in $06, $18 und $19 kommt man damit auf die Breite der Gesamtgrafik in Bytes.
$1A,$1B	Angabe, wie viele Zeilen der Gesamtgrafik bis zur Ausgabe im Fenster übersprungen werden sollen. Der untere Rand ergibt sich automatisch aus den Angaben in $07.

C2AD FindBAMBit

Testet, ob ein bestimmter Block der Diskette in der BAM schon als belegt gekennzeichnet ist. In $0E, $0F müssen dazu der Track und Sektor des Blocks enthalten sein. Ist der Akku anschließend gleich Null, so ist der Block bereits belegt.

C2B0 SetDevice

Freigabe des seriellen IEC-Busses für ein beliebiges Gerät. Im Akku muß die gewünschte Gerätenummer enthalten sein (z.B. Nummer 4, falls der Drucker angesprochen werden soll). Falls das Original-Floppy-DOS angesprochen werden soll, muß vorher noch PurgeTurbo = $C235 aufgerufen werden. Durch InitForIO = $C25C werden anschließend die 64er-Kernal-Routinen bereitgestellt. Nun können die gewünschten Bus-Aktionen stattfinden. Im Anschluß daran muß das GEOS-Kernal durch den Aufruf von DoneWithIO = $C25F wieder eingeblendet werden. Der Floppy-Speeder wird bei der Benutzung von High-Level-Routinen (z.B. GetFile, SaveFile) automatisch aktiviert. Lediglich bei den Routinen ReadBlock, WriteBlock und VerWrBlock muß er durch EnterTurbo = $C214 "von Hand" wieder gestartet werden.

Rückgabe: Error-Status im X-Register siehe Anhang

C2B3	IsMseInRegion

Mit dieser Routine kann getestet werden, ob sich die Maus derzeit in einem bestimmten Bildschirmbereich befindet.

Parameter:

$06	Obere Begrenzung (0-199)
$07	Untere Begrenzung (größer als in $06, max. 199)
$08,$09	Linke Begrenzung (0-319)
$10,$11	Rechte Begrenzung (größer als in $08/$09, max. 199)

Falls sich die Maus in dem angegebenen Bereich befindet, wird im Akku $FF übergeben, andernfalls $00.

C2B6	ReadByte

Gehört zu den VLIR-Routinen (siehe ab $C274). Holt ein Byte nach dem anderen aus aktuellem VLIR-Record in den Akku.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C2B9	FreeBlock (ab GEOS V1.3)

Gibt einen in der BAM als belegt gekennzeichneten Block frei.

Parameter

$0E,$0F Track und Sektor des freizugebenden Blocks.

In der GEOS-Version 1.2 ist FreeBlock noch nicht in der Sprungtabelle enthalten (drei NOP-Befehle). Man kann aber die Funktion durch einen direkten Einsprung ins Kernal aufrufen.

Die Adresse hierfür lautet $9844. Die Parameterübergabe ändert sich gegenüber V1.3 nicht.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C2BC	**ChangeDiskDevice**

Mit dieser Routine kann die Gerätenummer der derzeit aktuellen Floppy softwaremäßig auf einen anderen Wert gebracht werden. Hierzu muß ChangeDiskDevive die Nummer (8-11) im Akku übergeben werden.

Rückgabe: Error-Status im X-Register (siehe Anhang)

C2BF	**RstrFrmDialog**

Wird beim Anklicken der Icons einer Dialogbox (z.B. OK, CANCEL etc.) automatisch aufgerufen. Hierdurch wird das Fenster abgebaut, die eingefrorenen und geretteten Jobs werden reaktiviert (Pull-Down-Menüs, Prozesse etc.), und in $02 die Nummer des angeklickten Icons übergeben. Falls das Fenster jedoch von einer durch die Dialogbox aufgerufenen Unterroutine manuell abgebaut werden muß (siehe DoDlgBox= $C256), kann vor dem Aufruf von RstrFrmDialog in $02 ein beliebiger Wert übergeben werden, der zur Auswertung an das Hauptprogramm weitergegeben wird.

C2C2	**Panic**

Ausgabe von SYSTEM ERROR IN $XXXX. Die Routine zieht sich die Adresse vom Stack, bei der der Prozessor in ein BRK gelaufen ist, und wandelt sie in eine vierstellige ASCII-Zahl um.

Diese Zahl wird in einem Fenster als Abschiedsmitteilung ausgegeben.

C2C5	BitOtherClip

BitOtherClip besitzt eine ähnliche Funktion wie BitmapClip = $C2AA, bietet jedoch den Vorteil, daß die Grafikdaten nicht direkt im Speicher vorliegen müssen. Statt dessen wird BitOtherClip die Einsprungadresse einer Routine mitgeteilt, die bei jedem Aufruf das nächste Byte der komprimierten Grafik (dazu siehe BitmapUp = $C142) in den Akuu holt. Diese Routine wird in den meisten Fällen die Grafikdaten von der Floppy holen (z.B. mittels ReadByte = $C2B6). Sie darf dabei die Speicherstellen $02 bis $1D nicht überschreiben. Außer den schon bei der Beschreibung von BitmapClip angegebenen Positionsangaben erwartet BitOtherClip noch die folgenden Informationen:

$02,$03	Zeiger auf 134 Bytes umfassenden Datenpuffer.
$1C,$1D	Adresse der oben beschriebenen Routine
$1E,$1F	Adresse einer Routine, die den Zeiger in $02,$03 wieder auf den Anfang des Datenpuffers setzt.

Anhang C.1: Speicherbelegungsplan

Die folgende Tabelle zeigt Ihnen die Belegung des Speichers im Commodre 64. Ein besonderes Augenmerk bei der Durchsicht sollten Sie auf die durch "unbenutzt" gekennzeichneten Bereiche richten.

Diese Speicherstellen sind vom Betriebssystem oder vom BASIC-Interpreter nicht belegt und stellen daher - nicht nur für Assembler-Programmierer - einen idealen Datenspeicher dar.

Die Belegung der Zeropage

Hexadresse	Dezimal	Belegung
00	0	Datenrichtungs-Register für Prozessor-Port Bit 0 - 6; 0= Eingang 1= Ausgang
01	1	Im Prozessor-Port kann man angeben, welche Speicherbereiche ein- oder ausgeschaltet werden. Bit-Beschreibung: Bit 0 1 = BASIC-ROM, 0= RAM Bit 1 1 = Kernal-ROM, 0= RAM Bit 2 1 = I/O 0= Zeichensatz Bit 3 = Datenausgabe von Datasette Bit 4 0 = Taste bei Datasette gedrückt 1 = nicht gedrückt Bit 5 1 = Motor an, 0= Motor aus Die Bits 6 und 7 sind unbenutzt und immer 0
02	2	Unbenutzt
03-04	3-4	Vektor für Umwandlung von Fließkomma nach Fest Von diesen Adressen aus beginnt der Interpreter, eine Gleitkomma-Zahl in eine ganze Zahl umzuwandeln. Der Vektor deutet auf die Adresse $B1AA.
05-06	5-6	Vektor für Umwandlung von Fest nach Fließkomma Diese Routine verwandelt eine ganze Zahl in eine Fließkomma-Zahl. Der Zeiger steht auf $B391.
07		Suchzeichen Die Speicherzelle $07 wird oft von BASIC-Programmen als Suchzeiger für Texteingaben verwendet.

Hexadresse	Dezimal	Belegung
08	8	Hochkomma-Flag Wärend der Umwandlung von BASIC-Befehlen in "Tokens" wird die Speicherzelle $08 als Zwischenspeicher für BASIC-Texteingaben verwendet.
09	9	Speicher für Spalte beim TAB-Befehl Nach der Ausführung von TAB oder SPC wird die Cursor-Position in der Speicherzelle 9 zwischengespeichert.
0A	10	0= LOAD, 1= Verify, Flag des Interpreters Weil die Routine von LOAD und VERIFY identisch ist, wird ein Flag benötigt, um zu unterscheiden, ob ein LOAD- oder ein VERIFY-Vorgang ausgeführt worden ist.
0B	11	Zeiger im Eingabepuffer, Anzahl der Dimensionen Die Speicherzelle $0B wird dazu verwendet, die Anzahl der Dimensionen zu berechnen. Außerdem wird noch die Länge der "Token-Zeile" hier angegeben.
0C	12	Flag für DIM Diese Speicherzelle wird benutzt, um festzustellen, ob die Variable ein Array oder schon eine dimensionierte Variable ist.
0D	13	Typ-Flag $00= numerisch, $FF= String Das Flag zeigt dem BASIC-Interpreter an, ob es sich um Zahlenwerte oder um einen String handelt.
0E	14	$80= Integer, $00= Real Wenn eine Gleitkommazahl auftritt, steht in der Speicherzelle $00, bei einer ganzen Zahl eine $80.
0F	15	Hochkomma-Flag bei LIST Duch diese Speicherzelle wird beim LIST-Befehl duch ein Hochkomma erkannt, ob eine Textkette folgt. Zusätzlich wird in dieser Speicherzelle markiert, ob eine "Garbage Collection" durch- geführt werden muß oder nicht.
10	16	Flag für FN Hier wird angezeigt, ob es sich um eine Array-Variable oder um eine mit DEF FN definierte Variable handelt.

Hexadresse	Dezimal	Belegung
11	17	$00= INPUT, $40= GET, $98= READ Diese Speicherzelle gibt an, in welche Routine der BASIC-Interpreter verzweigen soll.
12	18	Vorzeichen bei ATN Die Speicherzelle $12 wird von den trigonometrischen Funktionen zur Bestimmung des Vorzeichens verwendet. Zusätzlich dient die Speicherzelle $12 als Vergleichsoperator für Vergleichsoperationen.
13	19	Aktives I/O-Gerät $00= Direkteingabe Die Speicherzelle $13 wird als Zeiger für die Peripheriegeräte, wie Tastatur, Datasette, RS232, User-Port, Bildschirm, Drucker und Floppy verwendet.
14-15	20-21	Integer-Adresse z.B. Zeilennummer In dieser Speicherzelle werden die Zeilennummern von den Befehlen, wie ON..GOTO, GOTO, GOSUB, ON..GOSUB und der Zeilenausgabe beim LIST-Befehl gespeichert.
16	22	Zeiger auf String-Stack Die Speicherzelle $16 zeigt auf den nächsten freien Speicherplatz im String-Stack.
17-18	23-24	Zeiger auf zuletzt verwendeten String Der Inhalt dieser beiden Bytes zeigt auf den zuletzt verwendeten Speicherplatz.
19-21	25-33	String-Stack Die Angaben im String-Stack enthalten die String-Länge sowie die Anfangs- und Endadressen des vorherigen Strings.
22-25	34-37	Zeiger für diverse Zwecke Diese Speicherzellen benutzt der Interpreter, um verschiedene Zwischenergebnisse zu speichern.
26-2A	38-42	Register für Funktionsauswertung und Arithmetik Diese Speicherzellen werden vom BASIC-Interpreter auch zum Speichern von Zwischenergebnissen bei der Multiplikation und Division benutzt.

Hexadresse	Dezimal	Belegung
2B-2C	43-44	Zeiger auf BASIC-Programmanfang Der Anfangsbereich des BASIC ist in Low- und High-Byte angegeben. Man kann durch die beiden Bytes den BASIC-Start abfragen oder verändern.
2D-2E	45-46	Zeiger auf BASIC-Programmende Dieser Zeiger teilt dem Interpreter das BASIC-Ende mit, damit die Variablen hinter dem Programm abgelegt werden können.
2F-30	47-48	Zeiger auf Start der Arrays Das Low- und High-Byte der Adressen geben dem BASIC-Interpreter die Information, ab welcher Speicherzelle die Arrays eines BASIC-Programms gespeichert sind.
31-32	48-50	Zeiger auf Ende der Datenfelder Diese beiden Speicherzellen zeigen auf das Ende der Arrays. Zu beachten ist, daß die Zeichenketten rückwärts gespeichert werden.
33-34	51-52	Zeiger auf String-Grenze Der Inhalt dieser Speicherzellen zeigt auf das Ende des Textspeichers, der aber noch zugleich das obere Ende des frei verfügbaren RAM-Bereichs anzeigt.
35-36	53-54	Hilfszeiger für Strings In diesen Zellen wird die Adresse der Zeichenkette verzeichnet, die als letzte von Routinen zur String-Manipulation abgespeichert worden ist.
37-38	55-56	Zeiger auf BASIC-RAM-Ende Dieser Zeiger gibt dem Interpreter an, welches die höchste von BASIC verwendbare Speicheradresse ist.
39-3A	57-58	Augenblickliche BASIC-Zeilennummer In diesen Speicherzellen wird die Zeilennummer verzeichnet, welche gerade ausgeführt wird.

Hexadresse	Dezimal	Belegung
3B-3C	59-60	Zeilennummer für CONT Falls eine Unterbrechung des Programmablaufs duch den Befehl STOP oder über die <Stop>-Taste erfolgt, wird in den Speicheradressen $3B-$3C die Zeilennummer gespeichert, die gerade abgearbeitet wurde.
3D-3E	61-62	Zeiger auf nächstes Statement für CONT Sobald eine neue BASIC-Zeile verarbeitet wird, holt sich das Betriebssystem die aktuelle Zeilennummer und speichert diese dann in $3D-$3E als Low- und High-Byte ab.
3F-40	63-64	Augenblickliche Zeilennummer für DATA Diese beiden Speicherzellen enthalten die Zeilennummer einer DATA-Zeile, die gerade vom READ-Befehl ausgelesen wird.
41-42	65-66	Zeiger für nächstes DATA-Element Hier ist die Adresse aufgeführt, ab welcher der READ-Befehl nach der nächsten DATA-Zeile sucht.
43-44	67-68	Zeiger auf Herkunft der Eingabe Der Zeiger zeigt auf die jeweilige Adresse in diesem Eingabepufferspeicher.
45-46	69-70	Variablenname Falls während des Ablaufs eines Programms eine Variable auftaucht, wird deren Name hier zwischengespeichert.
47-48	71-72	Variablenadresse In diesen Speicherzellen wird der Zeiger auf den Variablenwert abgelegt.
49-4A	73-74	Zeiger auf Variablenelement Die Adresse einer Schleifenvariable wird zunächst hier gespeichert, bevor sie in den Stack gebracht wird.
4B-4C	75-76	Zwischenspeicher für Programmzeiger Diese Speicherzellen dienen als Zwischenspeicher für mathematische Operationen. Außerdem werden die Speicherzellen auch noch vom READ-Befehl als Zwischenspeicher verwendet.

Hexadresse	Dezimal	Belegung
4D	77	Maske für Vergleichsoperationen Dieser Zeiger wird von mathematischen Routinen als Vergleichsoperator verwendet, daß heißt, um festzustellen, ob ein Wert kleiner, gleich oder größer ist.
4E-4F	78-79	Zeiger für FN In $4E-$4F ist die Adresse angegeben, wo die Variablen und ihre Werte abgelegt sind.
50-53	80-83	String-Descriptor In diesen Speicherzellen wird unter anderem die Schrittweite für "Garbage Collection" und andere wichtige Informationen für den Interpreter festgelegt.
54	84	Konstante $4C JMP für Funktionen Hier ist die Konstante für JMP ($4C) festgelegt.
55-56	85-86	Sprungvektor für Funktionen In $55-$56 werden die Sprungvektoren für die Funktionen angegeben.
57-5B	87-91	Register für Arithmetik, Akku #3 Die Register werden für die Zwischenspeicherung von Polynomauswertungen (TAN) benötigt.
5C-60	92-96	Register für Arithmetik, Akku #4, siehe oben
61-65	97-101	Fließkomma-Akku #1, FAC Diese Register werden für die Berechnung von Fließkomma-Zahlen verwendet.
66	102	Vorzeichen von FAC Der Zeiger gibt an, ob der Wert, der im FAC steht, positiv oder negativ ist.
67	103	Zähler für Polynomauswertung Diese Speicherzelle dient als Zähler für die Polynomauswertung.
68	104	Rundungs-Byte für FAC Hier wird angegeben, ob der Wert, der im FAC steht, auf- oder abgerundet werden soll.

Hexadresse	Dezimal	Belegung
69-6D	105-109	Fließkomma-Akku#2, ARG Diese Register werden für die Berechnung von Fließkomma-Zahlen verwendet.
6E	110	Vorzeichen von ARG Hier wird angegeben, ob der Wert, der im ARG steht, positiv oder negativ ist.
6F	111	Vergleichs-Byte der Vorzeichen von FAC und ARG Diese Speicherzelle gibt dem Interpreter an, ob die Vorzeichen der beiden Akkus übereinstimmen.
71-72	113-114	Zeiger für Polynomauswertung Hier ist in Low- und High-Byte angegeben, was ausgewertet werden soll.
73-8A	115-138	CHRGET-Routine Holt ein Zeichen aus dem BASIC-Text.
7A-7B	122-123	Programmzeiger In diesen Speicherzellen wird in Low- und High-Byte die Anfangsadresse des als *nächstes auszuführenden* Befehls im BASIC-RAM angegeben.
7C-8A	124-138	Unbenutzt
8B-8F	139-143	Letzter RND-Wert In diesen Registern wird der letzte RND-Wert im Fließkommaformat abgelegt.
90	144	Statuswort ST In dieser Speicherzelle, die auch mit der BASIC-Variable ST identisch ist, sind die Fehlermeldungen der Datasette und der Floppy verzeichnet: Datasette: Bit 0 = Unbenutzt Bit 1 = Unbenutzt Bit 2 = Kurzer Block Bit 3 = Langer Block Bit 4 = Lesefehler Bit 5 = Prüfsummenfehler Bit 6 = File-Ende Bit 7 = Band-Ende Floppy: Bit 0 = Fehler beim Schreiben Bit 1 = Fehler beim Lesen

Hexadresse	Dezimal	Belegung
		Bit 2 = Unbenutzt Bit 3 = Unbenutzt Bit 4 = Unbenutzt Bit 5 = Unbenutzt Bit 6 = Daten-Ende Bit 7 = DEVICE NOT PRESENT ERROR
91	145	Flag für <Stop>-Taste In dieser Speicherzelle wird vermerkt, ob die <Stop>-Taste gedrückt worden ist oder nicht.
92	146	Zeitkonstante für Band Dieses Register hat die Aufgabe, kleine Unterschiede bei der Aufnahmegeschwindigkeit auszugleichen.
93	147	Flag für LOAD $00, oder für VERIFY $01 Dieses Flag dient dem Betriebssystem dazu, um zu unterscheiden, ob eine LOAD- oder eine VERIFY-Operation erfolgt.
94	148	Flag bei IEC-Ausgabe Diese Adresse setzt bei Floppy und Drucker den "LISTEN" Zustand.
95	149	Ausgabepuffer für IEC-Bus Hier wird das Zeichen abgelegt, welches über den seriellen Port zur Floppy oder zum Drucker geschickt werden soll, sobald die Adresse $94 Bereitschaft zeigt.
96	150	Flag für EOT vom Band empfangen In $96 werden die Daten zwischengespeichert, die vom Band gelesen werden.
97	151	Zwischenspeicher für Register Beim Lesen von Band wird hier das X-Register zwischengespeichert.
98	152	Anzahl der offenen Files In dieser Speicherzelle wird festgehalten, wie viele Files gleichzeitig geöffnet sind.

Hexadresse	Dezimal	Belegung
99	153	Aktives Eingabegerät In dieser Speicherzelle wird festgehalten, welches Gerät zur Eingabe verwendet werden soll. Die Nummern sind folgendermaßen festgelegt: 0 = Tastatur 1 = Datasette 2 = RS232 und User-Port 3 = Bildschirm 4-5 = Drucker 8-11 = Laufwerke
9A	154	Aktives Ausgabegerät Diese Speicherzelle ist mit der vorherigen zu vergleichen, nur steht hier die Nummer des Geräts, über das die Ausgabe erfolgt.
9B	155	Parität für Band Über diese Speicherzelle findet eine Parity-Prüfung (Quersummenbildung) statt. Dies dient dazu, um Lese- und Schreibfehler zu vermeiden.
9C	156	Flag für Byte empfangen Hier wird festgelegt, ob das gelesene Byte die Quersumme richtig gebildet hat oder nicht.
9D	157	Flag für Direktmodus $80, Programm $00 In dieser Speicherzelle wird angegeben, welche Fehlermeldungen zugelassen werden und welche nicht. $00 unterdrückt alle Fehlermeldungen, $80 kommt dem normalen Eingabemodus gleich und $C0 läßt alle Fehlermeldungen zu. Diese Zustände können alle künstlich erzeugt werden.
9E	158	Band-Pass 1 Checksumme Diese Speicherzelle wird zur Überprüfung von Bytes bei Kassettenoperationen benutzt.
9F	159	Band-Pass 2 Fehlerkorrektur Hier werden die Fehler korrigiert, die bei Kassettenoperationen aufgetreten sind.

Hexadresse	Dezimal	Belegung
A0-A2	160-162	Time In diesen Speicherzellen wird die Uhrzeit über die Interruptroutine erhöht.
A3	163	Bit-Zähler für serielle Ausgabe Dieses Register wird als Zwischenspeicher von Ein-/Ausgaberoutinen benutzt.
A4	164	Zähler für Band siehe oben
A5	165	Zähler für Band schreiben Diese Speicherzelle wird als Zähler des Synchron-Bits verwendet.
A6	166	Zeiger in Bandpuffer Dieses Register wird als Zähler benutzt, welcher angibt, wie viele Bytes aus dem Bandpuffer gelesen oder in den Bandpuffer geschrieben worden sind.
A7-AB	167-171	Arbeitsspeicher für Ein-/Ausgabe Diese Register werden häufig von Kassettenoperationen und der RS-232 Schnittstelle als Zwischenspeicher benutzt.
AC-AD	172-173	Zeiger für Bandpuffer und Scrolling Diese Speicherzellen dienen in erster Linie als Zeiger auf die Adresse, ab welcher ein Programm geladen oder gespeichert werden soll. Zweitens dienen sie auch als Zwischenspeicher während des Scrollings des Video-RAM und beim Einfügen zusätzlicher Zeilen.
AE-AF	174-175	Zeiger auf Programmende bei LOAD/SAVE Ähnlich wie $AC-$AD funktionieren diese beiden Speicherzellen. Sie Zeigen immer auf das Byte, das gerade gelesen oder gespeichert wurde.
B0-B1	176-177	Der Wert in den Speicherzellen wird benutzt, um die Zeitkonstante beim Lesen von Band einzustellen.

Hexadresse	Dezimal	Belegung
B2-B3	178-179	Zeiger auf Bandpuffer Diese beiden Speicherzellen zeigen auf den Bandpuffer ($033C)
B4	180	Bit-Zähler für Band Hier wird die Anzahl der übertragenden Bits gezählt.
B5	181	Nächstes Bit für RS-232 Diese Speicherzelle enthält immer das nächste Bit, das bei RS-232 Operationen übertragen werden soll.
B6	182	Puffer für auszugebendes Byte Dieses Register wird als Ausgabezwischenspeicher benutzt.
B7	183	Länge des File-Namens In dieser Speicherzelle wird angegeben, aus wie vielen Zeichen der File-Name besteht.
B8	184	Logische File-Nummer In dieser Speicherzelle wird die logische File-Nummer verzeichnet.
B9	185	Sekundäradresse Hier steht die jeweilige Sekundäradresse.
BA	186	Gerätenummer Entsprechend ist auch in dieser Speicherzelle die Gerätenummer zu finden.
BB-BC	187-188	Zeiger auf File-Namen In diesen Speicherzellen steht ein Zeiger, der in Low- und High-Byte-Darstellung auf den File-Namen zeigt.
BD	189	Arbeitsspeicher serielle Ein-/Ausgabe Hier wird von den RS-232-Routinen ein Prüf-Byte abgelegt (Parity-Prüfung).
BE	190	Paß-Zähler für Band In dieser Speicherzelle ist angegeben, wie viele Blockteile von Band gelesen oder auf Band geschrieben werden sollen.

Hexadresse	Dezimal	Belegung
BF	191	Puffer für serielle Ausgabe Beim Laden eines Programms von Band wird diese Speicherzelle dazu benutzt, um die einzelnen Bits zu einem Byte zusammenzusetzen.
C0	192	Flag für Bandmotor Der Motor der Datasette kann nur eingeschaltet werden, wenn die Speicherzelle ungleich Null ist.
C1-C2	193-194	Startadresse für Ein-/Ausgabe In diesen Registern ist in Low- und High-Byte-Darstellung angegeben, ab welcher Adresse ein Programm geladen oder gespeichert wird.
C3-C4	195-196	Endadresse für Ein-/Ausgabe Hier steht in Low- und High-Byte der Zeiger auf den Tape-Header im Bandpuffer.
C5	197	Nummer der gedrückten Taste Hier wird die Nummer der gedrückten Taste gespeichert (64= keine Taste).
C6	198	Anzahl der gedrückten Tasten Hier steht die jeweilige Anzahl der Zeichen, die im Tastaturpuffer gespeichert sind.
C7	199	Flag für RVS-Modus Diese Speicherzelle gibt an, ob die auszugebenden Zeichen "revers" oder "normal" dargestellt werden sollen (0= normal, 1= revers).
C8	200	Zeilenende für Eingabe Dieses Register enthält die Position des letzen Zeichens in einer Zeile.
C9	201	Cursor-Zeile für Eingabe Diese Speicherzelle dient dazu, um die Zeile des letzten eingegebenen Zeichens festzustellen.
CA	202	Cursor-Spalte für Eingabe Diese Speicherzelle dient dazu, um die Spalte des letzten eingegebenen Zeichens festzustellen.

Hexadresse	Dezimal	Belegung
CB	203	Gedrückte Taste Hier steht der jeweilige Code der gedrückten Taste (64= keine Taste).
CC	204	Flag für Cursor Der Cursor wird ausgeschaltet, wenn in dieser Speicherzelle ein größerer Wert als Null steht.
CD	205	Zähler für Cursor blinken Diese Speicherzelle dient als Zähler für die Cursor-Blinkphase. Wenn der Wert 20 in dieser Speicherzelle abgezählt ist, wird der Cursor eingeschaltet.
CE	206	Zeichen unter dem Cursor Hier ist jeweils der Bildschirmcode eines Zeichens angegeben, das sich gerade unter dem Cursor befindet.
CF	207	Flag für Cursor In diesem Register wird festgehalten, in welcher Blink-Phase sich der Cursor gerade befindet.
D0	208	Flag für Eingabe von Tastatur oder Bildschirm Hier wird die Länge der zu übertragenden Zeichen gespeichert.
D1-D2	209-210	Zeiger auf Start der Bildschirmzeile In diesen Speicherzellen wird in Low- und High-Byte-Darstellung angezeigt, wo sich im Video-RAM die Zeile befindet, auf der der Cursor gerade steht.
D3	211	Cursor-Spalte Cursor-Spaltenposition
D4	212	Flag für Hochkomma-Modus Falls in dieser Speicherzelle eine Null steht, dann befindet sich der Computer im Hochkomma-Modus. Andere Werte bewirken den Normalmodus.
D5	213	Länge der Bildschirmzeile Der Inhalt dieser Speicherzelle entscheidet, ob eine neue Zeile angefangen werden muß oder nicht.

Hexadresse	Dezimal	Belegung
D6	214	Cursor-Zeile Hier wird die Zeilenposition des Cursors festgehalten.
D7	215	Speicher für ASCII-Tastencode Bevor ein Zeichen in den Tastaturpuffer gebracht wird, wird es vorher hier zwischengespeichert.
D8	216	Anzahl der Inserts Hier wird die Anzahl der Inserts festgelegt.
D9-F2	217-242	MSB der Bildschirmzeilenanfänge Alle 25 Speicherzellen enthalten Informationen über die Zeilen des Bildschirms.
F3-F4	243-244	Zeiger in Farb-RAM Diese Speicherzellen zeigen auf die Stelle im Farb-RAM, an der der Cursor auf der Zeile steht.
F5-F6	245-246	Zeiger auf Tastatur-Dekodiertabelle Diese Speicherzellen zeigen auf die Tastatur-Dekodiertabelle.
F7-F8	247-248	Zeiger auf RS-232-Eingabepuffer Diese Register zeigen auf die Anfangsadresse des Eingabepuffers.
F9-FA	249-250	Zeiger auf RS-232-Ausgabepuffer Diese Register zeigen auf die Anfangsadresse des Ausgabepuffers.
00FF-010A	255-266	Puffer für Umwandlung Fließkomma nach ASCII Diese Register werden für die Zwischenspeicherung von Fließkomma-zahlen benutzt.
0100-013E	256-318	Speicher für Korrektur bei Bandeingabe Beim Laden von Band werden hier die Daten zwischengespeichert, aus denen das Betriebssystem erkennen kann, welche Bytes fehlerhaft sind.
013F-01FF	256-511	Prozessor-Stack Der Stack ist generell ein Zwischenspeicher, in dem der Programmierer Daten ablegen kann. Außerdem wird er vom Prozessor dazu

Hexadresse	Dezimal	Belegung
		benutzt, bei einem Interrupt oder einem Unterprogramm-Aufruf die Adresse, von der aus verzweigt wurde, zwischenzuspeichern. Dies geschieht in der Reihenfolge High- und Low-Byte. Die Daten werden im Stack von der Adresse $01FF zur Adresse $0100 hin abgelegt. Bei einem BREAK wird zusätzlich der Prozessorstatus im Stack abgelegt.
0200-0258	512-600	BASIC-Eingabepuffer Nach der Eingabe eines Befehls oder einer Programmzeile werden diese Daten in diesen Bereich zwischengespeichert, um dann wieder weiterverarbeitet zu werden.
0259-0262	601-610	Tabelle der logischen File-Nummern In dieser Tabelle werden die logischen File-Nummern der Reihe nach, von 1-10, eingetragen. Beim Schließen einer Datei werden diese Einträge wieder entfernt.
0263-026C	611-620	Tabelle der Gerätenummern Diese Tabelle entspricht $0259-$0262, nur mit dem Unterschied, daß hier die Geräteadressen vermerkt werden.
026D-0276	621-630	Tabelle der Sekundäradressen Diese Tabelle entspricht $0259-$0262, nur mit dem Unterschied, daß hier die Sekundäradressen vermerkt werden.
0277-0280	631-640	Tastaturpuffer Hier werden die Tastencodes zwischengespeichert, die nicht sofort vom Betriebssystem weiterverarbeitet werden können.
0281-0282	641-642	Start des BASIC-RAM Nach einem RESET oder einem Kaltstart wird dieser Zeiger auf den nächsten freien Speicherplatz gesetzt.
0283-0284	643-644	Ende des BASIC-RAM Dieser Zeiger wird nach einem RESET oder einem Kaltstart auf den letzten verfügbaren freien RAM-Speicherplatz gesetzt.

Hexadresse	Dezimal	Belegung
0285	645	Timeout-Flag für seriellen IEC-Bus Alle Zähler in dieser Speicherzelle, die größer als 128 sind, bedeuten, daß ein Gerät angeschlossen ist. Kleinere Werte bedeuten das Gegenteil.
0286	646	Augenblickliche Farbe Hier wird die Zeichenfarbe festgelegt: 0 = schwarz 8 = orange 1 = weiß 9 = braun 2 = rot 10 = hellrot 3 = lila 11 = dunkelgrau 4 = purpur 12 = mittelgrau 5 = grün 13 = hellgrün 6 = blau 14 = hellblau 7 = gelb 15 = hellgrau
0287	647	Farbe unter dem Cursor In dieser Speicherzelle merkt sich das Betriebssystem, welche Farbe gerade unter dem Cursor steht.
0288	648	High-Byte Video-RAM Dieses High-Byte gibt dem Betriebssystem an, ab welcher Adresse das Video-RAM zu finden ist.
0289	649	Länge des Tastaturpuffers Dieses Register gibt an, wie viele Speicherzellen des Tastaturpuffers belegt werden sollen.
028A	650	Flag für Repeat-Funktion für alle Tasten In dieser Speicherzelle wird dem Betriebssystem angegeben, welche Tasten eine Repeat-Funktion haben und welche nicht: 0 = nur Cursor-, <Inst/Del>- und Leertaste 64 = keine Taste 128 = alle Tasten
028B	651	Zähler für Repeat-Geschwindigkeit Diese Speicherzelle dient als Zähler, die die Repeat-Geschwindigkeit festlegt.
028C	652	Zähler für Repeat-Verzögerung Hier wird angegeben, wie lange eine Taste gedrückt sein muß, bis die Repeat-Funktion einsetzt.

Hexadresse	Dezimal	Belegung
028D	653	Flag für <Shift>, <Commodore> und <Ctrl> In diesem Register stehen die Tastencodes der Steuertasten: 1 = <Shift> 2 = <Commodore> 3 = <Shift> und <Commodore> 4 = <Ctrl> 5 = <Shift> und <Ctrl> 6 = <Commodore> und <Ctrl> 7 = <Shift>, <Commodore> und <Ctrl>
028E	654	<Shift>-Flag Hier steht die zuletzt gedrückte Steuertaste.
028F-0290	655-656	Zeiger für Tastatur-Dekodierung Hier steht ein Zeiger, der auf die Betriebssystem-Routine für die Tastatur-Dekodierung zeigt.
0291	657	Flag für <Shift>/<Commodore> gesperrt Falls in der Speicherzelle eine 128 steht, wird eine Umschaltung mit <Shift>/<Commodore> verriegelt. Bei einer 0 wird die Umschaltung zugelassen.
0292	658	Flag für Scrollen Wenn in dieser Speicherzelle eine 0 steht, setzt der Scroll-Vorgang ein. Bei einem größeren Wert setzt dieser Vorgang nicht ein.
0293	659	RS-232 Kontrollwert Hier wird die Übertragungsgeschwindigkeit der RS-232 Schnittstelle festgelegt: Bits 0-3 steuern die Übertragungsgeschwindigkeit: Bit-Muster Wert Baud-Rate 0000 0 50 Baud 0001 1 50 Baud 0010 2 75 Baud 0011 3 110 Baud 0100 4 134.5 Baud 0101 5 150 Baud 0110 6 300 Baud 0111 7 600 Baud 1000 8 1200 Baud 1001 9 1800 Baud 1010 10 2400 Baud Bit 4 nicht belegt

Hexadresse	Dezimal	Belegung
		Die Bits 5 und 6 steuern die Länge der Übertragung: Bit-Muster Wert Länge 00 0 8-Bit 01 32 7-Bit 10 64 6-Bit 11 96 5-Bit Bit 7 gibt die Anzahl der STOP-Bits an: Bitmuster Wert Anzahl 0 0 1-STOP-Bit 1 128 2-STOP-Bits
0294	660	RS-232-Befehlswort Die Bits steuern das Handshake-Protokoll.
0295-0296	661-662	Bit-Timing Die Möglichkeit, eine frei wählbare Übertragungsgeschwindigkeit einzustellen, wurde vorgesehen, aber nicht eingebaut.
0297	663	RS-232-Status Hier werden die Fehlermeldungen der RS-232-Schnittstelle angezeigt: Bit Wert Bedeutung 0 1 Fehler bei Parity-Prüfung 1 2 Fehler in der Bit-Folge 2 4 Überlauf des Eingabepuffers 3 8 Eingabepuffer ist leer 4 16 Das CTS-Signal fehlt 5 32 Nicht belegt 6 64 Das DSR-Signal fehlt 7 128 Die Übertragung ist unterbrochen
0298	664	Anzahl der Daten-Bits für RS-232 Diese Speicherzelle wird verwendet, um die Wortlänge festzustellen.
0299-029A	665-666	RS-232-Baud-Rate Die Übertragungsrate errechnet sich aus der Systemfrequenz (985.25) KHz dividiert durch die Baud-Rate. Dieser Wert steht in Low- und High-Byte-Darstellung in den beiden Speicherzellen. Er wird vom Betriebssystem abgerufen.

Hexadresse	Dezimal	Belegung
029B	667	Zeiger für empfangenes Byte RS-232 Wenn man den Inhalt der Speicherzelle mit dem Wert in $F7-$F8 addiert, erhält man die Adresse des zuletzt im Eingabepuffer eingegebenen Bytes.
029C	668	Zeiger auf Input von RS-232 Wenn man den Inhalt der Speicherzelle mit dem Wert in $F7-$F8 addiert, erhält man die Adresse des ersten im Eingabepuffer eingegebenen Bytes.
029D	669	Zeiger auf zu übertragendes Byte RS-232 Wenn man den Inhalt der Speicherzelle mit dem Wert in $F9-$FA addiert, erhält man die Adresse des ersten im Ausgabepuffer eingegebenen Bytes.
029E	670	Zeiger auf Ausgabe auf RS-232 Wenn man den Inhalt der Speicherzelle mit dem Wert in $F9-$FA addiert, erhält man die Adresse des zuletzt im Ausgabepuffer eingegebenen Bytes.
029F-02A0	671-672	Speicher für IRQ während Bandbetrieb Bei Kassettenoperationen wird hier in Low- und High-Byte-Darstellung der Vektor für die Interruptroutine gespeichert.
02A1	673	CIA-2-NMI-Flag Diese Speicherzelle erhält den Wert des Interrupt-Steuer-Registers, das die RS-232-Schnittstelle steuert.
02A2	674	CIA-1-Timer A Bei Bandroutinen wird hier das High-Byte von Timer A zwischengespeichert.
02A3	675	CIA-1-Interrupt-Flag Bei Bandroutinen wird in dieser Speicherzelle festgelegt, welche IRQ freigegeben sind und welche nicht.
02A4	676	CIA-1-Flag für Timer A Hier wird bei Bandroutinen angegeben, ob Timer A läuft oder nicht. Wenn hier eine $00 steht, ist der Timer freigegeben, andernfalls ist er gesperrt.

Hexadresse	Dezimal	Belegung
02A5	677	Bildschirmzeile
02A6	678	Flag für PAL-(1) oder NTSC-Version (0) Hier steht ein Wert, der angibt, ob es sich um eine PAL- oder eine NTSC-Version handelt.
02C0-02FE	704-766	Sprite 11
0300-0301	768-769	$E38B Vektor für BASIC-Warmstart
0302-0303	770-771	$A483 Vektor für Eingabe einer Zeile
0304-0305	772-773	$A57C Vektor für Umwandlung in Interpretercode
0306-0307	774-775	$A71A Vektor für Umwandlung in Klartext (LIST)
0308-0309	776-777	$A7E4 Vektor für "BASIC-Befehlsadresse holen"
030A-030B	778-779	$AE86 Vektor für "Ausdruck auswerten"
030C	780	Akku für SYS-Befehl
030D	781	X-REG für SYS-Befehl
030E	782	Y-REG für SYS-Befehl
0310	783	Status-Register für SYS-Befehl
0311-0312	785-786	$B248 USR-Vektor
0314-0315	788-789	$EA31 IRQ-Vektor
0316-0317	790-791	$FE66 BRK-Vektor
0318-0319	792-793	$FE47 NMI-Vektor
031A-031B	794-795	$F34A OPEN-Vektor
031C-031D	796-797	$F291 CLOSE-Vektor
031E-031F	798-799	$F20E CHKIN-Vektor
0320-0321	800-801	$F250 CKOUT-Vektor
0322-0323	802-803	$F333 CLRCH-Vektor
0324-0325	804-805	$F157 INPUT-Vektor

Hexadresse	Dezimal	Belegung
0326-0327	806-807	$F1CA OUTPUT-Vektor
0328-0329	808-809	$F6ED STOP-Vektor
032A-032B	810-811	$F13E GET-Vektor
032B-032C	812-813	$F32F CLALL-Vektor
032E-032F	814-815	$FE66 Warmstart-Vektor
0330-0331	816-817	$F4A5 LOAD-Vektor
0332-0333	818-819	$F5ED SAVE-Vektor
033C-03FB	828-1019	Bandpuffer
0340-037E	832-894	Sprite 13
0380-03BE	896-958	Sprite 14
03C0-03FE	960-1022	Sprite 15

Anhang C.2: Die Register des VIC-Chips

Die nachfolgende Tabelle enthält eine komplette Übersicht über die Register des Video-Chips VIC. Die Basisadresse des VIC ist 53.248 ($D000). In der Tabelle finden Sie jeweils die Register-Nummer. Die tatsächliche Speicheradresse der einzelnen Register errechnet sich leicht wie folgt:

```
53.248 + Register-Nummer ($D000 + Register-Nummer)
```

Register	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00/$00		X-Position Sprite 0						
01/$01		Y-Position Sprite 0						
02/$02		X-Position Sprite 1						
03/$03		Y-Position Sprite 1						
04/$04		X-Position Sprite 2						
05/$05		Y-Position Sprite 2						
06/$06		X-Position Sprite 3						
07/$07		Y-Position Sprite 3						
08/$08		X-Position Sprite 4						
09/$09		Y-Position Sprite 4						
10/$0A		X-Position Sprite 5						
11/$0B		Y-Position Sprite 5						
12/$0C		X-Position Sprite 6						
13/$0D		Y-Position Sprite 6						
14/$0E		X-Position Sprite 7						
15/$0F		Y-Position Sprite 7						
16/$10	Spr7	Spr6	Spr5	Spr4	Spr3	Spr2	Spr1	Spr0
		High-Byte X-Position						
17/$11	High- Byte	Hin- ter- grund	Schalt-Bits Hi- res	Bild- sch irm	Zei- len anz	Wert der Zeilen- verschiebung in Y-Richtung Smooth Scroll.		
18/$12	Raster-Register +High-Byte in Register 17, Bit7							
19/$13		Lichtgriffel X-Position						
20/$14		Lichtgriffel Y-Position						
21/$15	Spr7	Spr6	Spr5	Spr4	Spr3	Spr2	Spr1	Spr0
	Ein-/Ausschalten von Sprites (0=aus, 1=ein)							
22/$16	nicht benutzt		RESET Bit	Mehr- farb- modus	Spal- ten- zahl	Wert der Spalten- verschiebung in X-Richtung		
23/$17	Spr7	Spr6	Spr5	Spr4	Spr3	Spr2	Spr1	Spr0
	Sprite-Vergrößerung Y-Richt. 0=normal 1=doppelt							
24/$18	Startadr. Textbildsch.							

Register	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
25/$19	Interrupt-Flags				Licht	Spr	Hin-	Ras-
					grif-	Kolli-	ter-	ter
26/$1A	Interrupt-Masken				fel	sion	grund	
27/$1B	Spr7	Spr6	Spr5	Spr4	Spr3	Spr2	Spr1	Spr0
	Sprite-Hintergrund-Prior. 0=Sprite 1=Hintergr.							
28/$1C	Sprite-Mehrfarbmodus 0=normal 1=mehrfarbig							
29/$1D	Spritevergrößerung X-Richt. 0=normal 1=doppelt							
30/$1E	Sprite/Sprite-Kollision 0 = nein 1=ja							
31/$1F	Sprite/Hintergrund-Kollision 0=nein 1=ja							
32/$20	nicht		benutzt		Bildschirmrahmenfarbe			
33/$21	nicht		benutzt		Hintergrundfarbe 0			
34/$22	nicht		benutzt		Hintergrundfarbe 1			
35/$23	nicht		benutzt		Hintergrundfarbe 2			
36/$24	nicht		benutzt		Hintergrundfarbe 3			
37/$25	nicht		benutzt		Sprite-Mehrfarben 0			
38/$26	nicht		benutzt		Sprite-Mehrfarben 1			
39/$27	nicht		benutzt		Farbe von Sprite 0			
40/$28	nicht		benutzt		Farbe von Sprite 1			
41/$29	nicht		benutzt		Farbe von Sprite 2			
42/$2A	nicht		benutzt		Farbe von Sprite 3			
43/$2B	nicht		benutzt		Farbe von Sprite 4			
44/$2C	nicht		benutzt		Farbe von Sprite 5			
45/$2D	nicht		benutzt		Farbe von Sprite 6			
46/$2E	nicht		benutzt		Farbe von Sprite 7			

Anhang C.3: Bildschirm-, Hires-Grafik- und Sprite-Speicherbereiche

Der Speicher des Commodore 64 wird vom Video-Chip VIC in vier 16-KByte-Blöcke unterteilt. Da der VIC nur 16 KByte adressieren kann, müssen sich alle Grafikdaten (Bildschirm-, Hires-Bitmap usw.) jeweils zusammen in einem der vier Blöcke befinden. In den folgenden vier Tabellen finden Sie die möglichen Startadressen der einzelnen Grafik-Datenbereiche. Zur Erinnerung: Ein Bildschirmspeicher umfaßt ein KByte; ein Sprite-Zeigerbereich acht Byte und eine Hires-Bitmap acht KByte.

Bildschirmspeicher	Sprite-Zeiger	Hires-Bitmap
VIC-Block 0: 00000-16383 ($0000-$3FFF)		******************
1024 ($0400)	2040 ($07F8)	8192 ($2000)
2048 ($0800)	3064 ($0BF8)	8192 ($2000)
3072 ($0C00)	4088 ($0FF8)	8192 ($2000)
8192 ($2000)	9208 ($23F8)	—
9216 ($2400)	10232 ($27F8)	—
10240 ($2800)	11256 ($2BF8)	—
11264 ($2C00)	12280 ($2FF8)	—
12288 ($3000)	13304 ($33F8)	—
13312 ($3400)	14328 ($37F8)	—
14336 ($3800)	15352 ($3BF8)	—
15360 ($3C00)	16376 ($3FF8)	—
VIC-Block 1: 16384-32767 ($4000-$7FFF)		******************
16384 ($4000)	17400 ($43F8)	24576 ($6000)
17408 ($4400)	18424 ($47F8)	24576 ($6000)
18432 ($4800)	19448 ($4BF8)	24576 ($6000)
19456 ($4C00)	20472 ($4FF8)	24576 ($6000)
20480 ($5000)	21496 ($53F8)	24576 ($6000)
21504 ($5400)	22520 ($57F8)	24576 ($6000)
22528 ($5800)	23544 ($5BF8)	24576 ($6000)
23552 ($5C00)	24568 ($5FF8)	24576 ($6000)
24576 ($6000)	25592 ($63F8)	16384 ($4000)
25600 ($6400)	26616 ($67F8)	16384 ($4000)

Bildschirmspeicher	Sprite-Zeiger	Hires-Bitmap
26624 ($6800)	27640 ($6BF8)	16384 ($4000)
27648 ($6C00)	28664 ($6FF8)	16384 ($4000)
28672 ($7000)	29688 ($73F8)	16384 ($4000)
29696 ($7400)	30712 ($77F8)	16384 ($4000)
30720 ($7800)	31736 ($7BF8)	16384 ($4000)
31744 ($7C00)	32760 ($7FF8)	16384 ($4000)
VIC-Block 2: 32768-49151 ($8000-$BFFF) ******************		
32768 ($8000)	33784 ($83F8)	40960 ($A000)
33792 ($8400)	34808 ($87F8)	40960 ($A000)
34816 ($8800)	35832 ($8BF8)	40960 ($A000)
35840 ($8C00)	36856 ($8FF8)	40960 ($A000)
————	————	32768 ($8000)
VIC-Block 3: 49152-65535 ($C000-$FFFF) ******************		
49152 ($C000)	50168 ($C3F8)	57344 ($E000)
50176 ($C400)	51192 ($C7F8)	57344 ($E000)
51200 ($C800)	52216 ($CBF8)	57344 ($E000)
52224 ($CC00)	53240 ($CFF8)	57344 ($E000)
————	————	49152 ($C000)

Anhang C.4: Die Register des SID-Chips

Die Basisadresse des SID ist 54.272 ($D400). In der Tabelle finden Sie jeweils die Register-Nummer. Die tatsächliche Speicheradresse der einzelnen Register errechnet sich leicht wie folgt:

```
54.272 + Register-Nummer ($D400 + Register-Nummer)
```

Register		Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
00/$00	S	F r e q u e n z (Low-Wert)								
01/$01	t	F r e q u e n z (High-Wert)								
02/$02	i	P u l s w e i t e (Low-Wert)								
03/$03	m	--	--	--	--	Pulsweite (high)				
04/$04	m	Rau	Recht	Säge	Drei	Test	Ring	Syn	Gate	S
	e	schen	eck	zahn	eck		mod.	chron		c
05/$05		A t t a c k				D e c a y				h
06/$06	1	S u s t a i n				R e l e a s e				r
										e
07/$07	S	F r e q u e n z (Low-Wert)								i
08/$08	t	F r e q u e n z (High-Wert)								b
09/$09	i	P u l s w e i t e (Low-Wert)								r
10/$0A	m	--	--	--	--	Pulsweite (high)				e
11/$0B	m	Rau	Recht	Säge	Drei	Test	Ring	Syn	Gate	g
	e	schen	eck	zahn	eck		mod.	chron		i
12/$0C		A t t a c k				D e c a y				s
13/$0D	2	S u s t a i n				R e l e a s e				t
										e
14/$0E	S	F r e q u e n z (Low-Wert)								r
15/$0F	t	F r e q u e n z (High-Wert)								
16/$10	i	P u l s w e i t e (Low-Wert)								
17/$11	m	--	--	--	--	Pulsweite (high)				
18/$12	m	Rau	Recht	Säge	Drei	Test	Ring	Syn	Gate	
	e	schen	eck	zahn	eck		mod.	chron		
19/$13		A t t a c k				D e c a y				
20/$14	3	S u s t a i n				R e l e a s e				
21/$15	F	--	--	--	--	--	Grenzfreq. (low)			
22/$16	i	G r e n z f r e q u e n z (high)								
23/$17	l	R e s o n a n z				F i l t e r				
24/$18	t	Aus	Hoch	Band	Tief	L a u t s t ä r k e				
	.		pass	pass	pass					
25/$19	R	P o t e n t i o m e t e r X								L
26/$1A	e	P o t e n t i o m e t e r Y								e
27/$1B	g	O s z i l l a t o r Stimme 3								s
28/$1C	.	H ü l l k u r v e Stimme 3								e

Anhang C.5: Musiknoten und zugehörige Frequenzen

Die folgende Tabelle enthält sämtliche auf dem Commodore 64 spielbaren Musiknoten und ihre zugehörigen Frequenzen (in Hertz). In den Spalten stehen jeweils alle Noten einer Oktave.

Zwei Beispiele: Der Note "g#" in der 3. Oktave entspricht die Frequenz 207.7 Hertz, der Note "d" in der 7. Oktave die Frequenz 2349.3 Hertz.

	0	1	2	3	4	5	6	7
c	16.4	32.7	65.4	130.8	261.6	523.3	1046.5	2093.0
c#	17.3	34.6	69.3	138.6	277.2	554.4	1108.7	2217.5
d	18.4	36.7	73.4	146.8	293.7	587.3	1174.7	2349.3
d#	19.4	38.9	77.8	155.6	311.1	622.3	1244.5	2489.0
e	20.6	41.2	82.4	164.8	329.6	659.3	1318.5	2637.0
f	21.8	43.7	87.3	174.6	349.2	698.5	1396.9	2793.8
f#	23.1	46.2	92.5	185.0	370.0	740.0	1480.0	2960.0
g	24.5	49.0	98.0	196.0	392.0	784.0	1568.0	3136.0
g#	26.0	51.9	103.8	207.7	415.3	830.6	1661.2	3322.4
a	27.5	55.0	110.0	220.0	440.0	880.0	1760.0	3520.0
a#	29.1	58.3	116.5	233.1	466.2	932.3	1864.7	3729.3
h	30.9	61.7	123.5	246.9	493.9	987.8	1975.5	——

Anhang C.6: Adressen wichtiger BASIC-2.0-Routinen

Im folgenden finden Sie eine Auswahl interessanter Routinen des BASIC-2.0-Interpreters.

BASIC-Kaltstartvektor ($E394)	$A000
BASIC-Warmstartvektor ($E37B)	$A002
Blockverschieberoutine	$A3B8
Prüft auf Platz im Stapelspeicher	$A3FB
Prüft auf Platz im Speicher	$A408
Ausgabe von OUT OF MEMORY ERROR	$A435
Fehlermeldung ausgeben	$A437
Einsprung für BREAK	$A469
Einsprung für READY	$A474
Eingabe-Warteschleife	$A480
Löschen und Einfügen von Programmzeilen	$A49C
BASIC-Programmzeilen neu binden	$A533
Holt eine BASIC-Zeile in den Eingabepuffer	$A560
Ausgabe von STRING TOO LONG ERROR	$A571
Umwandlung einer Zeile in Interpretercode	$A579
Startadresse einer BASIC-Zeile suchen	$A613
BASIC-Befehl NEW	$A642
BASIC-Befehl CLR	$A65E
Programmzeiger auf BASIC-Start setzen	$A68E
BASIC-Befehl LIST	$A69C
Token in Klartext umwandeln	$A717
BASIC-Befehl FOR	$A742
Zentrale Interpreter-Schleife	$A7AE
BASIC-Befehl ausführen	$A7ED
BASIC-Befehl RESTORE	$A81D
Bricht Programm bei gedrückter <Stop>-Taste ab	$A82C
BASIC-Befehl STOP	$A82F
BASIC-Befehl END	$A831
BASIC-Befehl CONT	$A857
BASIC-Befehl RUN	$A871
BASIC-Befehl GOSUB	$A883
BASIC-Befehl GOTO	$A8A0
BASIC-Befehl RETURN	$A8D2
BASIC-Befehl DATA	$A8F8
Nächstes Statement suchen	$A906
Nächste BASIC-Zeile suchen	$A909
BASIC-Befehl IF	$A928
BASIC-Befehl REM	$A93B
BASIC-Befehl ON	$A94B
Adresse einer BASIC-Zeile suchen	$A96B
BASIC-Befehl LET	$A9A5
BASIC-Befehl PRINT#	$AA80
BASIC-Befehl CMD	$AA86
BASIC-Befehl PRINT	$AAA0
String ausgeben	$AB1E
Leerzeichen bzw. CURSOR RIGHT ausgeben	$AB3E
BASIC-Befehl GET	$AB7B

Anhang C.7: Adressen wichtiger Kernal-Routinen

Im folgenden finden Sie eine Auswahl interessanter Betriebssystem-Routinen.

IRQ-Interrupt-Routine	$EA31
RESET-Routine	$FCE2
NMI-Interrupt-Routine	$FE43
Video-RESET	$FF81
CIA initialisieren	$FF84
RAM löschen und testen	$FF87
I/O initialisieren	$FF8A
I/O-Vektoren initialisieren	$FF8D
Status setzen	$FF90
Sekundäradresse nach LISTEN senden	$FF93
Sekundäradresse nach TALK senden	$FF96
RAM-Ende setzen oder holen	$FF99
RAM-Anfang setzen oder holen	$FF9C
Tastatur abfragen	$FF9F
Timeout-Flag für IEC-Bus setzen	$FFA2
Eingabe vom IEC-Bus	$FFA5
Ausgabe auf IEC-Bus	$FFA8
UNTALK senden	$FFAB
UNLISTEN senden	$FFAE
LISTEN senden	$FFB1
TALK senden	$FFB4
Status holen	$FFB7
File-Parameter setzen	$FFBA
File-Namen-Parameter setzen	$FFBD
OPEN-Routine	$FFC0
CLOSE-Routine	$FFC3
Eingabegerät setzen	$FFC6
Ausgabegerät setzen	$FFC9
Zeicheneingabe	$FFCF
Zeichenausgabe	$FFD2
LOAD-Routine	$FFD5
SAVE-Routine	$FFD8
Time setzen	$FFDB
Time holen	$FFDE
<Stop>-Taste abfragen	$FFE1
GET-Routine	$FFE4
CLALL-Routine	$FFE7
Time erhöhen	$FFEA
Cursor setzen/Cursor-Position holen	$FFF0

Anhang C.8: Alphabetisches Verzeichnis der ROM-Routinen

Alphabetisches Verzeichnis der ROM-Routinen

BASIC-Befehl ON $A94B

Plus FAC = Konstante (A/Y) + FAC $B867
Polynomberechnung 1 .... $E043
Polynomberechnung 2 .... $E059
Positive Integer-Zahl in A/X ausgeben .... $BDCD
Potenzierung FAC = ARG hoch FAC $BF7B
Potenzierung FAC = ARG hoch Konstante (A/Y) $BF78
Programm vom Band laden $F84A
Programm-Header vom Band lesen $F72C
Programmzeiger auf BASIC-Start $A68E
Programmzeile einfügen $A4ED
Programmzeile löschen $A4A9
Programmzeilen neu binden $A533
Prüfung auf numerisch $AD8D
Prüfung auf Auto-Start-ROM $FD02
Prüfung auf Buchstabe $B113
Prüfung auf Erreichen der Endadresse $FCD1
Prüfung auf Klammer auf .... $AEFA
Prüfung auf Klammer zu $AEF7
Prüfung auf Komma .... $AEFD
Prüfung auf Platz im Speicher $A3FB
Prüfung auf <Shift>, <Ctrl>, <Commodore> $EB48
Prüfung auf Steuerzeichen $EC44
Prüfung auf <Stop>-Taste $A82C
Prüfung auf String $AD8F
Prüfung auf Systemvariable $AF14
Prüfung auf Übereinstimmung mit laufendem Zeichen .... $AEFD
Rechtsverschieben eines Registers $B983
Rekordermotor ausschalten .... $FCCA
RESET-Routine $FCE2
RAM für BASIC initialisieren $E3BF
ROM-Modul-Identifizierung $FD10
RS-232-Ausgabe .... $EEBB
RS-232-Ausgabe $F208
RS-232-CHKIN $F04D
RS-232-GET $F086
Schafft Platz im Speicher $A408
Sekundäradresse nach Listen senden $EDB9
Sekundäradresse nach Talk senden $EDC7
Stapelsuch-Routine $A38E
Startadresse des Bandpuffers holen $F7D0
Startadresse einer Programmzeile berechnen $A613
Status holen $FE07
<Stop>-Taste abfragen $F6ED
String ausgeben $AB1E
String holen, Zeiger nach A/Y $B487
String in reservierten Bereich übertragen $B67A
String-Parameter holen $B782
String-Parameter vom Stack holen $B761
String-Vergleich $B02E
String-Verknüpfung $B63D
String-Verwaltung, FRESTR $B6A3
String-Zeiger berechnen $B475
SAVE-Routine $F5DD

Anhang C.9: Commodore-64-ROM-Listing A000-AFFF

```
******************************
A000 94 E3                    Start-Vektor  $E394
A002 7B E3                    NMI-Vektor    $E37B

******************************
A004 43 42 4D 42 41 53 49 43  'cbmbasic'

****************************** Adressen der BASIC-Befehle -1
                              Interpretercode Adresse Befehl
A00C 30 A8               $80    $A831    END
A00E 41 A7               $81    $A742    FOR
A010 1D AD               $82    $AD1E    NEXT
A012 F7 A8               $83    $A8F8    DATA
A014 A4 AB               $84    $ABA5    INPUT#
A016 BE AB               $85    $ABBF    INPUT
A018 80 B0               $86    $B081    DIM
A01A 05 AC               $87    $AC06    READ
A01C A4 A9               $88    $A9A5    LET
A01F 9F A8               $89    $A8A0    GOTO
A020 70 A8               $8A    $A871    RUN
A022 27 A9               $8B    $A928    IF
A024 1C A8               $8C    $A81D    RESTORE
A026 82 A8               $8D    $A883    GOSUB
A028 D1 A8               $8E    $A8D2    RETURN
A02A 3A A9               $8F    $A93B    REM
A02C 2E A8               $90    $A82F    STOP
A02F 4A A9               $91    $A94B    ON
A030 2C B8               $92    $B82D    WAIT
A032 67 E1               $93    $E168    LOAD
A034 55 E1               $94    $E156    SAVE
A036 64 E1               $95    $E165    VERIFY
A038 B2 B3               $96    $B3B3    DEF
A03A 23 B8               $97    $B824    POKE
A03C 7F AA               $98    $AA80    PRINT#
A03E 9F AA               $99    $AAA0    PRINT
A040 56 A8               $9A    $A857    CONT
A042 9B A6               $9B    $A69C    LIST
A044 5D A6               $9C    $A65E    CLR
A046 85 AA               $9D    $AA86    CMD
A048 29 E1               $9E    $E12A    SYS
A04A BD E1               $9F    $E1BE    OPEN
A04C C6 E1               $A0    $E1C7    CLOSE
A04E 7A AB               $A1    $AB7B    GET
A050 41 A6               $A2    $A642    NEW

****************************** Adressen der BASIC-Funktionen
A052 39 BC               $B4    $BC39    SGN
A054 CC BC               $B5    $BCCC    INT
A056 58 BC               $B6    $BC58    ABS
A058 10 03               $B7    $0310    USR
A05A 7D B3               $B8    $B37D    FRE
```

```
A05C 9E B3                  $B9     $B39E   POS
A05E 71 BF                  $BA     $BF71   SQR
A060 97 E0                  $BB     $E097   RND
A062 EA B9                  $BC     $B9EA   LOG
A064 ED BF                  $BD     $BFED   EXP
A066 64 E2                  $BE     $E264   COS
A068 6B E2                  $BF     $E26B   SIN
A06A B4 E2                  $C0     $E2B4   TAN
A06C 0E E3                  $C1     $E30E   ATN
A06E 0D B8                  $C2     $B80D   PEEK
A070 7C B7                  $C3     $B77C   LEN
A072 65 B4                  $C4     $B465   STR$
A074 AD B7                  $C5     $B7AD   VAL
A076 8B B7                  $C6     $B78B   ASC
A078 EC B6                  $C7     $B6EC   CHR$
A07A 00 B7                  $C8     $B700   LEFT$
A07C 2C B7                  $C9     $B72C   RIGHT$
A07E 37 B7                  $CA     $B737   MID$

****************************** Hierachiecodes und
                               Adressen-1 der Operatoren
A080 79 69 B8                  $79, $B86A   Addition
A083 79 52 B8                  $79, $B853   Subtraktion
A086 7B 2A BA                  $7B, $BA2B   Multiplikation
A089 7B 11 BB                  $7B, $BB12   Division
A08C 7F 7A BF                  $7F, $BF7B   Potenzierung
A08F 50 E8 AF                  $50, $AFE9   AND
A092 46 E5 AF                  $46, $AFE6   OR
A095 7D B3 BF                  $7D, $BFB4   Vorzeichenwechsel
A098 5A D3 AE                  $5A, $AED4   NOT
A09B 64 15 B0                  $64, $B016   Vergleich

****************************** BASIC-Befehlsworte
A09E 45 4E                       end
A0A0 C4 46 4F D2 4E 45 58 D4     for   next
A0A8 44 41 54 C1 49 4E 50 55     data  input#
A0B0 54 A3 49 4E 50 55 D4 44     input dim
A0B8 49 CD 52 45 41 C4 4C 45     read  let
A0C0 D4 47 4F 54 CF 52 55 CE     goto  run
A0C8 49 C6 52 45 53 54 4F 52     if    restore
A0D0 C5 47 4F 53 55 C2 52 45     gosub return
A0D8 54 55 52 CE 52 45 CD 53     rem   stop
A0E0 54 4F D0 4F CE 57 41 49     on    wait
A0E8 D4 4C 4F 41 C4 53 41 56     load  save
A0F0 C5 56 45 52 49 46 D9 44     verify def
A0F8 45 C6 50 4F 4B C5 50 52     poke  print#
A100 49 4E 54 A3 50 52 49 4E     print
A108 D4 43 4F 4E D4 4C 49 53     cont  list
A110 D4 43 4C D2 43 4D C4 53     clr   cmd   sys
A118 59 D3 4F 50 45 CE 43 4C     open  close
A120 4F 53 C5 47 45 D4 4E 45     get   new
A128 D7 54 41 42 A8 54 CF 46     tab(  to
A130 CE 53 50 43 A8 54 48 45     spc(  then
```

```
A138 CE 4E 4F D4 53 54 45 D0    not   stop
A140 AB AD AA AF DE 41 4E C4    + - * / ^  and
A148 4F D2 BE BD BC 53 47 CE    or <=> sgn
A150 49 4E D4 41 42 D3 55 53    int   abs   usr
A158 D2 46 52 C5 50 4F D3 53    fre   pos   sqr
A160 51 D2 52 4E C4 4C 4F C7    rnd   log
A168 45 58 D0 43 4F D3 53 49    exp   cos   sin
A170 CE 54 41 CE 41 54 CE 50    tan   atn   peek
A178 45 45 CB 4C 45 CE 53 54    len   str$
A180 52 A4 56 41 CC 41 53 C3    val   asc
A188 43 48 52 A4 4C 45 46 54    chr$  left$
A190 A4 52 49 47 48 54 A4 4D    right$   mid$
A198 49 44 A4 47 CF 00          go
***************************** BASIC-Fehlermeldungen
A19E 54 4F                      1  too many files
A1A0 4F 20 4D 41 4E 59 20 46
A1A8 49 4C 45 D3 46 49 4C 45    2  file open
A1B0 20 4F 50 45 CE 46 49 4C    3  file not open
A1B8 45 20 4E 4F 54 20 4F 50
A1C0 45 CE 46 49 4C 45 20 4E    4  file not found
A1C8 4F 54 20 46 4F 55 4E C4    5  device not present
A1D0 44 45 56 49 43 45 20 4E
A1D8 4F 54 20 50 52 45 53 45
A1E0 4E D4 4E 4F 54 20 49 4E    6  not input file
A1E8 50 55 54 20 46 49 4C C5
A1F0 4E 4F 54 20 4F 55 54 50    7  not output file
A1F8 55 54 20 46 49 4C C5 4D
A200 49 53 53 49 4E 47 20 46    8  missing File-Name
A208 49 4C 45 20 4E 41 4D C5
A210 49 4C 4C 45 47 41 4C 20    9  illegal device number
A218 44 45 56 49 43 45 20 4E
A220 55 4D 42 45 D2 4E 45 58    10 next without for
A228 54 20 57 49 54 48 4F 55
A230 54 20 46 4F D2 53 59 4E    11 syntax
A238 54 41 D8 52 45 54 55 52    12 return without gosub
A240 4E 20 57 49 54 48 4F 55
A248 54 20 47 4F 53 55 C2 4F    13 out of data
A250 55 54 20 4F 46 20 44 41
A258 54 C1 49 4C 4C 45 47 41    14 illegal quantity
A260 4C 20 51 55 41 4E 54 49
A268 54 D9 4F 56 45 52 46 4C    15 overflow
A270 4F D7 4F 55 54 20 4F 46    16 out of memory
A278 20 4D 45 4D 4F 52 D9 55    17 undef'd statement
A280 4E 44 45 46 27 44 20 53
A288 54 41 54 45 4D 45 4E D4
A290 42 41 44 20 53 55 42 53    18 bad subscript
A298 43 52 49 50 D4 52 45 44    19 redim'd array
A2A0 49 4D 27 44 20 41 52 52
A2A8 41 D9 44 49 56 49 53 49    20 division by zero
A2B0 4F 4E 20 42 59 20 5A 45
A2B8 52 CF 49 4C 4C 45 47 41    21 illegal direct
A2C0 4C 20 44 49 52 45 43 D4
A2C8 54 59 50 45 20 4D 49 53    22 type mismatch
```

```
A2D0 4D 41 54 43 C8 53 54 52    23 string too long
A2D8 49 4E 47 20 54 4F 4F 20
A2E0 4C 4F 4E C7 46 49 4C 45    24 file data
A2E8 20 44 41 54 C1 46 4F 52    25 formula too complex
A2F0 4D 55 4C 41 20 54 4F 4F
A2F8 20 43 4F 4D 50 4C 45 D8
A300 43 41 4E 27 54 20 43 4F    26 can't continue
A308 4E 54 49 4E 55 C5 55 4E    27 undef'd function
A310 44 45 46 27 44 20 46 55
A318 4E 43 54 49 4F CE 56 45    28 verify
A320 52 49 46 D9 4C 4F 41 C4    29 load

****************************** Adressen der Fehlermeldungen
A328 9E A1 AC A1 B5 A1 C2 A1
A330 D0 A1 E2 A1 F0 A1 FF A1
A338 10 A2 25 A2 35 A2 3B A2
A340 4F A2 5A A2 6A A2 72 A2
A348 7F A2 90 A2 9D A2 AA A2
A350 BA A2 C8 A2 D5 A2 E4 A2
A358 ED A2 00 A3 0E A3 1E A3
A360 24 A3 83 A3

****************************** Meldungen des Interpreters
A364 0D 4F 4B 0D                   OK
A368 00 20 20 45 52 52 4F 52       ERROR
A370 00 20 49 4E 20 00 0D 0A       IN
A378 52 45 41 44 59 2E 0D 0A       READY.
A380 00 0D 0A 42 52 45 41 4B       BREAK
A388 00 A0

****************************** Stapel-Suchroutine für
                               FOR-NEXT- und GOSUB-Befehl

Einsprung von $A8D8, $A749, $AD2B

A38A  BA        TSX         Stapelzeiger in X-Register
A38B  E8        INX         4 mal erhöhen
A38C  E8        INX         (nächsten zwei Rücksprung-
A38D  E8        INX         adressen, Interpreter und
A38E  E8        INX         Routine, übergehen)
A38F  BD 01 01  LDA $0101,X nächstes Byte holen
A392  C9 81     CMP #$81    Ist es FOR-Code ?
A394  D0 21     BNE $A3B7   Nein: dann RTS
A396  A5 4A     LDA $4A     Variablenzeiger holen
A398  D0 0A     BNE $A3A4   keine Variable (NEXT):$A3A4
A39A  BD 02 01  LDA $0102,X Variablenzeiger aus
A39D  85 49     STA $49     Stapel nach $49/4A
A39F  BD 03 01  LDA $0103,X (Variablenzeiger)
A3A2  85 4A     STA $4A     holen
A3A4  DD 03 01  CMP $0103,X Mit Zeiger im Stapel vergl.
A3A7  D0 07     BNE $A3B0   Ungleich: nächste Schleife
A3A9  A5 49     LDA $49     Zeiger wieder holen
A3AB  DD 02 01  CMP $0102,X Mit Zeiger im Stapel vergl.
```

```
A3AE    F0 07       BEQ $A3B7     Gleich: Schleife gefunden,RTS
A3B0    8A          TXA           Suchzeiger in Akku
A3B1    18          CLC           Carry für Addition löschen
A3B2    69 12       ADC #$12      Suchzeiger um 18 erhöhen
A3B4    AA          TAX           und wieder zurück ins X-Rg.
A3B5    D0 D8       BNE $A38F     nächste Schleife prüfen
A3B7    60          RTS           Rücksprung

****************************** Block-Verschieberoutine

Einsprung von $A749, $B15D

A3B8    20 08 A4    JSR $A408     prüft auf Platz im Speicher
A3BB    85 31       STA $31       Ende des Arraybereichs
A3BD    84 32       STY $32       als Beginn für freien Platz

Einsprung von $B628

A3BF    38          SEC           Carry löschen (Subtraktion)
A3C0    A5 5A       LDA $5A       Startadresse von Endad. des
A3C2    E5 5F       SBC $5F       Bereichs abziehen (Low)
A3C4    85 22       STA $22       Ergebnis (=Länge) speichern
A3C6    A8          TAY           Gleiches System für High:
A3C7    A5 5B       LDA $5B       Altes Blockende (High) und
A3C9    E5 60       SBC $60       davon alter Blockanfang sub
A3CB    AA          TAX           Länge nach X bringen
A3CC    E8          INX           Ist ein Rest ( Länge nicht
A3CD    98          TYA           256 Bytes)?
A3CE    F0 23       BEQ $A3F3     Nein: dann nur ganze Blöcke
A3D0    A5 5A       LDA $5A       Alte Endadresse (Low) und
A3D2    38          SEC           davon Länge des Restab-
A3D3    E5 22       SBC $22       schnitts subtrahieren ergibt
                                  Adresse des
A3D5    85 5A       STA $5A       Restabschnitts
A3D7    B0 03       BCS $A3DC     Berechnung für High umgehen
A3D9    C6 5B       DEC $5B       Dasselbe System für High
A3DB    38          SEC           Carry setzen (Subtraktion)
A3DC    A5 58       LDA $58       Alte Endadresse (High) und
A3DE    E5 22       SBC $22       davon Länge des Rests sub-
A3E0    85 58       STA $58       trahieren ergibt neue Adresse
A3E2    B0 08       BCS $A3EC     Unbedingter Sprung zur
A3E4    C6 59       DEC $59       Kopierroutine für ganze
A3E6    90 04       BCC $A3EC     Blöcke
A3E8    B1 5A       LDA ($5A),Y   Kopierroutine für Rest-
A3EA    91 58       STA ($58),Y   abschnitt
A3EC    88          DEY           Zähler vermindern
A3ED    D0 F9       BNE $A3E8     Alles? wenn nicht: weiter
A3EF    B1 5A       LDA ($5A),Y   Kopierroutine für ganze
A3F1    91 58       STA ($58),Y   Blöcke
A3F3    C6 5B       DEC $5B       Adreßzähler vermindern
A3F5    C6 59       DEC $59       Adreßzähler vermindern
A3F7    CA          DEX           Zähler vermindern
```

```
A3F8   D0 F2      BNE $A3EC   Alles? Wenn nicht: weiter
A3FA   60         RTS         sonst Rücksprung

****************************** Prüfung auf Platz im Stapel

Einsprung von $A885 ,$ADAE

A3FB   0A         ASL         Akku muß die halbe Zahl an
A3FC   69 3E      ADC #$3E    erforderlichem Platz haben
A3FE   B0 35      BCS $A435   gibt 'OUT OF MEMORY'
A400   85 22      STA $22     Wert merken
A402   BA         TSX         Ist Stapelzeiger kleiner
A403   E4 22      CPX $22     (2 * Akku + 62)?
A405   90 2E      BCC $A435   Wenn ja, dann OUT OF MEMORY
A407   60         RTS         Rücksprung

****************************** Schafft Platz im Speicher

Einsprung von $A3B8, $B264, $B2B9, $E426

A408   C4 34      CPY $34     für Zeileneinfügung
A40A   90 28      BCC $A434   und Variablen
A40C   D0 04      BNE $A412   A/Y = Adresse, bis zu der
A40E   C5 33      CMP $33     Platz benötigt wird.
A410   90 22      BCC $A434   Kleiner als Stringzeiger
A412   48         PHA         Akku zwischenspeichern
A413   A2 09      LDX #$09    Zähler setzen
A415   98         TYA         Y-Register auf
A416   48         PHA         Stapel retten
A417   B5 57      LDA $57,X   Ab $57 zwischenspeichern
A419   CA         DEX         Zähler vermindern
A41A   10 FA      BPL $A416   Alle? sonst weiter
A41C   20 26 B5   JSR $B526   Garbage Collection
A41F   A2 F7      LDX #$F7    Zähler setzen, um
A421   68         PLA         Akku, Y-Register und andere
A422   95 61      STA $61,X   Register zurückholen
A424   E8         INX         Zähler vermindern
A425   30 FA      BMI $A421   Fertig? Nein, dann weiter
A427   68         PLA         Y-Register von Stapel
A428   A8         TAY         zurückholen
A429   68         PLA         Akku holen
A42A   C4 34      CPY $34     Ist jetzt genügend Platz?
A42C   90 06      BCC $A434   Ja: dann Rücksprung
A42E   D0 05      BNE $A435   Kein Platz, dann Fehler-
A430   C5 33      CMP $33     meldung ' out of memory '
A432   B0 01      BCS $A435   ausgeben
A434   60         RTS         Rücksprung

Einsprung von $B30B

A435   A2 10      LDX #$10    Fehlernummer 'out of memory'
```

```
****************************** Fehlereinsprung

Einsprung von $A573, $A85F, $A8E5, $AB68, $AD32, $AD9B
$AF0A, $B24A, $B3B0, $B4D2, $B65A, $B980, $BB8C, $E109
$E19E

A437   6C 00 03   JMP ($0300) Zum BASIC-Warmstart ($E38B)

****************************** Fehlermeldung ausgeben

Einsprung von $E38E

A43A   8A         TXA         Fehlernummer im X-Register
A43B   0A         ASL         Akku * 2
A43C   AA         TAX         Akku als Zeiger nach X
A43D   BD 26 A3   LDA $A326,X und Adresse der
A440   85 22      STA $22     Fehlernummer aus Tabelle
A442   BD 27 A3   LDA $A327,X holen und
A445   85 23      STA $23     abspeichern
A447   20 CC FF   JSR $FFCC   I/O Kanäle zurücksetzen
A44A   A9 00      LDA #$00    und Eingabekanal auf
A44C   85 13      STA $13     Tastatur setzen
A44E   20 D7 AA   JSR $AAD7   (CR) und (LF) ausgeben
A451   20 45 AB   JSR $AB45   '?' ausgeben
A454   A0 00      LDY #$00    Zeiger setzen
A456   B1 22      LDA ($22),Y Fehlermeldungstext holen
A458   48         PHA         Akku retten
A459   29 7F      AND #$7F    Bit 7 löschen und
A45B   20 47 AB   JSR $AB47   Fehlermeldung ausgeben
A45E   C8         INY         Zähler vermindern
A45F   68         PLA         Akku zurückholen
A460   10 F4      BPL $A456   Fertig? Nein, dann weiter
A462   20 7A A6   JSR $A67A   BASIC-Zeiger initialisieren
A465   A9 69      LDA #$69    Zeiger A/Y auf Error-
A467   A0 A3      LDY #$A3    Meldung stellen

Einsprung von $A851

A469   20 1E AB   JSR $AB1E   String ausgeben
A46C   A4 3A      LDY $3A     Auf Programmodus
A46E   C8         INY         (prog/direkt) prüfen
A46F   F0 03      BEQ $A474   Direkt: dann ausgeben
A471   20 C2 BD   JSR $BDC2   'in Zeilennummer' ausgeben

Einsprung von $E391

A474   A9 76      LDA #$76    Zeiger auf Ready-Modus
A476   A0 A3      LDY #$A3    setzen und
A478   20 1E AB   JSR $AB1E   String ausgeben
A47B   A9 80      LDA #$80    Wert für Direktmodus laden
A47D   20 90 FF   JSR $FF90   und Flag setzen
```

```
****************************** Eingabe-Warteschleife

Einsprung von $A530

A480   6C 02 03   JMP ($0302) JMP $A483

Einsprung von $A480

A483   20 60 A5   JSR $A560   BASIC-Zeile nach
                              Eingabepuffer
A486   86 7A      STX $7A     CHRGET Zeiger auf
A488   84 7B      STY $7B     Eingabepuffer
A48A   20 73 00   JSR $0073   nächstes Zeichen holen
A48D   AA         TAX         Puffer leer?
A48E   F0 F0      BEQ $A480   Ja: dann weiter warten
A490   A2 FF      LDX #$FF    Wert für
A492   86 3A      STX $3A     Kennzeichen für Direktmodus
A494   90 06      BCC $A49C   Ziffer? als Zeile einfügen
A496   20 79 A5   JSR $A579   BASIC-Zeile in Code wandeln
A499   4C E1 A7   JMP $A7E1   Befehl ausführen

****************************** Löschen und Einfügen von
                              Programmzeilen
A49C   20 6B A9   JSR $A96B   Zeilenr. nach Adreßformat
A49F   20 79 A5   JSR $A579   BASIC-Zeile in Code wandeln
A4A2   84 0B      STY $0B     Zeiger in Eingabepuffer
A4A4   20 13 A6   JSR $A613   Zeilenadresse berechnen
A4A7   90 44      BCC $A4ED   Vorhanden? Ja: löschen
****************************** Programmzeile löschen
A4A9   A0 01      LDY #$01    Zeiger setzen
A4AB   B1 5F      LDA ($5F),Y Startadresse der nächsten
A4AD   85 23      STA $23     Zeile (High) setzen
A4AF   A5 2D      LDA $2D     Variablenanfangszeiger
A4B1   85 22      STA $22     (Low) setzen
A4B3   A5 60      LDA $60     Startadresse der zu
A4B5   85 25      STA $25     löschenden Zeile (High)
A4B7   A5 5F      LDA $5F     Startadresse der zu
A4B9   88         DEY         löschenden Zeile (Low)
A4BA   F1 5F      SBC ($5F),Y Startadresse der nächsten
A4BC   18         CLC         Zeile (Low)
A4BD   65 2D      ADC $2D     Variablenanfangszeiger (Low)
A4BF   85 2D      STA $2D     ergibt neuen Variablenan-
A4C1   85 24      STA $24     fangszeiger (Low)
A4C3   A5 2E      LDA $2E     Gleiches System für
A4C5   69 FF      ADC #$FF    High-Byte des Variablenan-
A4C7   85 2E      STA $2E     fangszeigers
A4C9   E5 60      SBC $60     minus Startadresse der zu
A4CB   AA         TAX         löschenden Zeile (Low) ergibt
A4CC   38         SEC         die zu verschiebenden Blöcke
A4CD   A5 5F      LDA $5F     Startadresse (Low) minus
A4CF   E5 2D      SBC $2D     Variablenanfangszeiger (Low)
A4D1   A8         TAY         ergibt Länge des Restabschn.
A4D2   B0 03      BCS $A4D7   Größer als 255? Nein: $A4D7
```

```
A4D4   E8         INX          Zähler für Blöcke erhöhen
A4D5   C6 25      DEC $25      Transportzeiger vermindern
A4D7   18         CLC          Carry löschen
A4D8   65 22      ADC $22      Anfangszeiger (Low)
A4DA   90 03      BCC $A4DF    Verminderung überspringen
A4DC   C6 23      DEC $23      Zeiger um 1 vermindern
A4DE   18         CLC          Carry löschen
A4DF   B1 22      LDA ($22),Y  Verschiebeschleife
A4E1   91 24      STA ($24),Y  Wert abspeichern
A4E3   C8         INY          Zähler um 1 erhöhen
A4E4   D0 F9      BNE $A4DF    Block fertig? Nein: weiter
A4E6   E6 23      INC $23      1.Adreßzeiger erhöhen (Low)
A4E8   E6 25      INC $25      2.Adreßzeiger erhöhen (Low)
A4EA   CA         DEX          Blockzähler um 1 vermindern
A4EB   D0 F2      BNE $A4DF    Alle Blöcke? Nein: weiter

******************************  Programmzeile einfügen
A4ED   20 59 A6   JSR $A659    CLR-Befehl
A4F0   20 33 A5   JSR $A533    Programmzeilen neu binden
A4F3   AD 00 02   LDA $0200    Zeichen im Puffer?
A4F6   F0 88      BEQ $A480    Nein: dann zur Warteschleife
A4F8   18         CLC          Carry löschen
A4F9   A5 2D      LDA $2D      Variablenanfangszeiger (Low)
A4FB   85 5A      STA $5A      als Endadresse (Quellbereich)
A4FD   65 0B      ADC $0B      + Länge der Zeile als End-
A4FF   85 58      STA $58      adresse des Zielbereichs Low
A501   A4 2E      LDY $2E      Variablenanfangszeiger als
A503   84 5B      STY $5B      Endadr. des Quellbereichs Low
A505   90 01      BCC $A508    Kein Übertrag? dann $A508
A507   C8         INY          Übertrag addieren
A508   84 59      STY $59      Als Endadresse
                               des Zielbereichs
A50A   20 B8 A3   JSR $A3B8    BASIC-Zeilen verschieben
A50D   A5 14      LDA $14      Zeilennummer aus
A50F   A4 15      LDY $15      $14/15 vor
A511   8D FE 01   STA $01FE    BASIC-Eingabepuffer setzen
A514   8C FF 01   STY $01FF    (ab $0200).
A517   A5 31      LDA $31      Neuer Variablen-
A519   A4 32      LDY $32      endzeiger
A51B   85 2D      STA $2D      als Zeiger auf Programm-
A51D   84 2E      STY $2E      ende speichern
A51F   A4 0B      LDY $0B      Zeilenlänge holen
A521   88         DEY          und um 1 vermindern
A522   B9 FC 01   LDA $01FC,Y  Zeile aus Eingabepuffer
A525   91 5F      STA ($5F),Y  ins Programm kopieren
A527   88         DEY          Schon alle Zeichen?
A528   10 F8      BPL $A522    Nein: dann weiterkopieren

Einsprung von $E1B2

A52A   20 59 A6   JSR $A659    CLR-Befehl
A52D   20 33 A5   JSR $A533    Programmzeilen neu binden
A530   4C 80 A4   JMP $A480    zur Eingabe-Warteschleife
```

```
****************************** BASIC-Zeilen neu binden

Einsprung von $A4F0, $A52D, $E1B8

A533   A5 2B      LDA $2B      Zeiger auf BASIC-Programm-

A535   A4 2C      LDY $2C      start holen und
A537   85 22      STA $22      und als Suchzeiger nach
A539   84 23      STY $23      $22/23 speichern
A53B   18         CLC          Carry löschen
A53C   A0 01      LDY #$01     Zeiger laden
A53E   B1 22      LDA ($22),Y  Zeilenadresse holen
A540   F0 1D      BEQ $A55F    =0? Ja: dann RTS
A542   A0 04      LDY #$04     Zeiger auf erstes BASIC-
A544   C8         INY          zeichen setzen
A545   B1 22      LDA ($22),Y  Zeichen holen
A547   D0 FB      BNE $A544    =0? (Zeilenende) Nein: weiter
A549   C8         INY          Zeilenlänge nach
A54A   98         TYA          Akku schieben
A54B   65 22      ADC $22      + Zeiger auf aktuelle Zeile
A54D   AA         TAX          (Low) ins X-Register
A54E   A0 00      LDY #$00     Zeiger laden
A550   91 22      STA ($22),Y  Akku als Adr.zeiger (Low)
A552   A5 23      LDA $23      Zeiger auf aktuelle
                               Zeile (High)
A554   69 00      ADC #$00     Übertrag addieren
A556   C8         INY          Zähler um 1 erhöhen
A557   91 22      STA ($22),Y  Adreßzeiger (High) speichern
A559   86 22      STX $22      Startadresse der nächsten
A55B   85 23      STA $23      Zeile abspeichern
A55D   90 DD      BCC $A53C    Zum Zeilenanfang
A55F   60         RTS          Rücksprung

****************************** Eingabe einer Zeile

Einsprung von $A483, $AC03

A560   A2 00      LDX #$00     Zeiger setzen
A562   20 12 E1   JSR $E112    ein Zeichen holen
A565   C9 0D      CMP #$0D     <Return>-Taste?
A567   F0 0D      BEQ $A576    Ja: dann Eingabe beenden
A569   9D 00 02   STA $0200,X  Zeichen nach Eingabepuffer
A56C   E8         INX          Zeiger um 1 erhöhen
A56D   E0 59      CPX #$59     89. Zeichen  ?
A56F   90 F1      BCC $A562    Nein: weitere Zeichen holen
A571   A2 17      LDX #$17     Nummer für 'string too long'
A573   4C 37 A4   JMP $A437    Fehlermeldung ausgeben
A576   4C CA AA   JMP $AACA    Puffer mit $0 abschließen, CR

****************************** Umwandlung einer Zeile in den
                               Interpreter-Code
```

Einsprung von $A496, $A49F

```
A579   6C 04 03   JMP ($0304) JMP $A57C
```

Einsprung von $A579

```
A57C   A6 7A      LDX $7A      Zeiger setzen, erstes Zeichen
A57E   A0 04      LDY #$04     Wert für codierte Zeile
A580   84 0F      STY $0F      Flag für Hochkomma
A582   BD 00 02   LDA $0200,X  Zeichen aus Puffer holen
A585   10 07      BPL $A58E    kein BASIC-Code ? kleiner 128
A587   C9 FF      CMP #$FF     Code für Pi ?
A589   F0 3E      BEQ $A5C9    Ja: dann speichern
A58B   E8         INX          Zeiger erhöhen
A58C   D0 F4      BNE $A582    nächstes Zeichen überprüfen
A58E   C9 20      CMP #$20     ' ' Leerzeichen?
A590   F0 37      BEQ $A5C9    Ja: dann speichern
A592   85 08      STA $08      in Hochkomma-Flag speichern
A594   C9 22      CMP #$22     '"' Hochkomma?
A596   F0 56      BEQ $A5EE    Ja: dann speichern
A598   24 0F      BIT $0F      Überprüft auf Bit 6
A59A   70 2D      BVS $A5C9    gesetzt: ASCII speichern
A59C   C9 3F      CMP #$3F     '?' Fragezeichen?
A59E   D0 04      BNE $A5A4    Nein: dann weiter prüfen
A5A0   A9 99      LDA #$99     PRINT-Code für ? laden
A5A2   D0 25      BNE $A5C9    und abspeichern
A5A4   C9 30      CMP #$30     Kleiner $30 ? (Code für 0)
A5A6   90 04      BCC $A5AC    Ja: dann $A5AC
A5A8   C9 3C      CMP #$3C     Mit $3C vergleichen
A5AA   90 1D      BCC $A5C9    wenn größer, dann $A5C9
A5AC   84 71      STY $71      Zeiger zwischenspeichern
A5AE   A0 00      LDY #$00     Zähler für Token-Tabelle
A5B0   84 0B      STY $0B      initialisieren
A5B2   88         DEY
A5B3   86 7A      STX $7A      Zeiger auf Eingabepuffer
A5B5   CA         DEX          zwischenspeichern
A5B6   C8         INY          X- und Y-Register
A5B7   E8         INX          um 1 erhöhen
A5B8   BD 00 02   LDA $0200,X  Zeichen aus Puffer laden
A5BB   38         SEC          Carry für Subtr. löschen
A5BC   F9 9E A0   SBC $A09E,Y  Zeichen mit Befehlswort vergleichen
A5BF   F0 F5      BEQ $A5B6    Gefunden? Ja: nächstes Zeich.
A5C1   C9 80      CMP #$80     mit $80 (128) vergleichen
A5C3   D0 30      BNE $A5F5    Befehl nicht gefunden: $A5F5
A5C5   05 0B      ORA $0B      BASIC-Code gleich Zähler +$80
A5C7   A4 71      LDY $71      Zeiger auf cod. Zeile holen
A5C9   E8         INX
A5CA   C8         INY          Zeiger erhöhen
A5CB   99 FB 01   STA $01FB,Y  BASIC-Code speichern
A5CE   B9 FB 01   LDA $01FB,Y  und Status-Register setzen
A5D1   F0 36      BEQ $A609    =0 (Ende): dann fertig
A5D3   38         SEC          Carry setzen (Subtraktion)
A5D4   E9 3A      SBC #$3A     ':' Trennzeichen?
```

```
A5D6   F0 04      BEQ $A5DC     Ja: dann $A5DC
A5D8   C9 49      CMP #$49      DATA-Code ?
A5DA   D0 02      BNE $A5DE     Nein: Speichern überspringen
A5DC   85 0F      STA $0F       nach Hochkomma-Flag speichern
A5DE   38         SEC           Carry setzen
A5DF   E9 55      SBC #$55      REM-Code ?
A5E1   D0 9F      BNE $A582     Nein: zum Schleifenanfang
A5E3   85 08      STA $08       0 in Hochkomma-Flag
A5E5   BD 00 02   LDA $0200,X   nächstes Zeichen holen
A5E8   F0 DF      BEQ $A5C9     =0 (Ende)? Ja: dann $A5C9
A5EA   C5 08      CMP $08       Als ASCII speichern?
A5EC   F0 DB      BEQ $A5C9     Nein: dann $A5C9
A5EE   C8         INY           Zeiger erhöhen
A5EF   99 FB 01   STA $01FB,Y   Code abspeichern
A5F2   E8         INX           Zeiger erhöhen
A5F3   D0 F0      BNE $A5E5     Zum Schleifenanfang
A5F5   A6 7A      LDX $7A       Zeiger wieder auf Eingabepuffer
A5F7   E6 0B      INC $0B       Suchzähler erhöhen
A5F9   C8         INY           Zähler erhöhen
A5FA   B9 9D A0   LDA $A09D,Y   nächsten Befehl suchen
A5FD   10 FA      BPL $A5F9     Gefunden? Nein: weitersuchen
A5FF   B9 9E A0   LDA $A09E,Y   Ende der Tabelle?
A602   D0 B4      BNE $A5B8     Nein: dann weiter
A604   BD 00 02   LDA $0200,X   nächstes Zeichen holen
A607   10 BE      BPL $A5C7     kleiner $80? Ja: $A5C7
A609   99 FD 01   STA $01FD,Y   im Eingabepuffer speichern
A60C   C6 7B      DEC $7B       CHRGET-Zeiger zurücksetzen
A60E   A9 FF      LDA #$FF      Zeiger auf Eingabepuffer -1
A610   85 7A      STA $7A       setzen (Low)
A612   60         RTS           Rücksprung

******************************  Startadresse einer
                                Programmzeile berechnen

Einsprung von $A4A4, $A6A7

A613   A5 2B      LDA $2B       Zeiger auf BASIC-
A615   A6 2C      LDX $2C       Programmstart laden

Einsprung von $A8C0

A617   A0 01      LDY #$01      Zähler setzen
A619   85 5F      STA $5F       BASIC-Programmstart als
A61B   86 60      STX $60       Zeiger nach $5F/60
A61D   B1 5F      LDA ($5F),Y   Link-Adresse holen (High)
A61F   F0 1F      BEQ $A640     gleich null: dann Ende
A621   C8         INY           Zähler 2 mal erhöhen ( Low-
A622   C8         INY           Byte übergehen)
A623   A5 15      LDA $15       gesuchte Zeilennummer (High)
A625   D1 5F      CMP ($5F),Y   mit aktueller vergleichen
A627   90 18      BCC $A641     kleiner: dann nicht gefunden
A629   F0 03      BEQ $A62E     gleich: Nummer Low prüfen
A62B   88         DEY           Zähler um 1 vermindern
```

```
A62C   D0 09      BNE $A637    unbedingter Sprung
A62E   A5 14      LDA $14      gesuchte Zeilennummer (Low)
A630   88         DEY          Zeiger um 1 vermindern
A631   D1 5F      CMP ($5F),Y  Zeilennummer Low vergleichen
A633   90 0C      BCC $A641    kleiner: Zeile nicht gefunden
A635   F0 0A      BEQ $A641    oder gleich: C=1 und RTS
A637   88         DEY          Y-Register auf 1 setzen
A638   B1 5F      LDA ($5F),Y  Adresse der nächsten Zeile
A63A   AA         TAX          in das X-Register laden
A63B   88         DEY          Register vermindern (auf 0)
A63C   B1 5F      LDA ($5F),Y  Link-Adresse holen (Low)
A63E   B0 D7      BCS $A617    weiter suchen
A640   18         CLC          Carry löschen
A641   60         RTS          Rücksprung

*****************************  BASIC-Befehl  NEW
A642   D0 FD      BNE $A641    Kein Trennzeichen: SYNTAX
                               ERROR

Einsprung von $E444

A644   A9 00      LDA #$00     Nullcode laden
A646   A8         TAY          und als Zähler ins Y-Reg.
A647   91 2B      STA ($2B),Y  Nullcode an Programmanfang
A649   C8         INY          Zähler erhöhen
A64A   91 2B      STA ($2B),Y  noch einen Nullcode dahinter
A64C   A5 2B      LDA $2B      Zeiger auf Programmst. (Low)
A64E   18         CLC          Carry löschen
A64F   69 02      ADC #$02     Programmstart + 2 ergibt
A651   85 2D      STA $2D      neuen Variablenstart (Low)
A653   A5 2C      LDA $2C      Zeiger auf Programmst. (High)
A655   69 00      ADC #$00     + Übertrag ergibt neuen
A657   85 2E      STA $2E      Variablenstart (High)

Einsprung von $A4ED, $A52A, $A87A

A659   20 8E A6   JSR $A68E    CHRGET, Routine neu setzen
A65C   A9 00      LDA #$00     Zero-Flag für CLR = 1 setzen

*****************************  BASIC-Befehl  CLR
A65E   D0 2D      BNE $A68D    Kein Trennzeichen: SYNTAX
                               ERROR

Einsprung von $A87D

A660   20 E7 FF   JSR $FFE7    alle I/O Kanäle zurücksetzen

Einsprung von $E101

A663   A5 37      LDA $37      Zeiger auf BASIC-RAM-Ende
A665   A4 38      LDY $38      (Low/High) laden
A667   85 33      STA $33      String-Start auf BASIC-
A669   84 34      STY $34      RAM-Ende setzen
```

```
A66B    A5 2D       LDA $2D      Zeiger auf Variablen-
A66D    A4 2E       LDY $2E      start laden
A66F    85 2F       STA $2F      und in Array-Anfangs-
A671    84 30       STY $30      zeiger setzen
A673    85 31       STA $31      und in Zeiger auf Array-
A675    84 32       STY $32      Ende speichern

Einsprung von $E1BB

A677    20 1D A8    JSR $A81D    RESTORE-Befehl

Einsprung von $A462, $E382

A67A    A2 19       LDX #$19     Wert laden und String-
A67C    86 16       STX $16      Descriptor-Index zurücksetzen
A67E    68          PLA          2 Bytes vom Stapel in das
A67F    A8          TAY          Y-Register und den
A680    68          PLA          Akku holen
A681    A2 FA       LDX #$FA     Wert laden und damit
A683    9A          TXS          Stapelzeiger initialisieren
A684    48          PHA          2 Bytes aus dem Y-Register
A685    98          TYA          und dem Akku wieder auf
A686    48          PHA          den Stapel schieben
A687    A9 00       LDA #$00     Wert laden und damit
A689    85 3E       STA $3E      CONT sperren
A68B    85 10       STA $10      und in FN-Flag speichern
A68D    60          RTS          Rücksprung

*****************************    Programmzeiger auf
                                 BASIC-Start

Einsprung von $A659, $E1B5

A68E    18          CLC          Carry löschen (Addition)
A68F    A5 2B       LDA $2B      Zeiger auf Programmstart (Low)
A691    69 FF       ADC #$FF     minus 1 ergibt
A693    85 7A       STA $7A      neuen CHRGET-Zeiger (Low)
A695    A5 2C       LDA $2C      Programmstart (High)
A697    69 FF       ADC #$FF     minus 1 ergibt
A699    85 7B       STA $7B      CHRGET-Zeiger (High)
A69B    60          RTS          Rücksprung

*****************************    BASIC Befehl  LIST
A69C    90 06       BCC $A6A4    Ziffer ? (Zeilennummer)
A69E    F0 04       BEQ $A6A4    nur LIST ?
A6A0    C9 AB       CMP #$AB     Code für '-'?
A6A2    D0 E9       BNE $A68D    anderer Code, dann SYNTAX ERR
A6A4    20 6B A9    JSR $A96B    Zeilennummer holen
A6A7    20 13 A6    JSR $A613    Startadresse berechnen
A6AA    20 79 00    JSR $0079    CHRGOT letztes Zeichen holen
A6AD    F0 0C       BEQ $A6BB    keine Zeilennummer
A6AF    C9 AB       CMP #$AB     Code für '-'?
A6B1    D0 8E       BNE $A641    Nein: SYNTAX ERROR
```

```
A6B3   20 73 00   JSR $0073    CHRGET nächstes Zeichen holen
A6B6   20 6B A9   JSR $A96B    Zeilennummer holen
A6B9   D0 86      BNE $A641    Kein Trennzeichen: SYNTAX ERR
A6BB   68         PLA          2 Bytes von Stapel holen
A6BC   68         PLA          (Rücksprungadresse übergehen)
A6BD   A5 14      LDA $14      zweite Zeilennummer laden
A6BF   05 15      ORA $15      gleich null ?
A6C1   D0 06      BNE $A6C9    Nein: $A6C9
A6C3   A9 FF      LDA #$FF     Wert laden und
A6C5   85 14      STA $14      zweite Zeilennummer Maximal-
A6C7   85 15      STA $15      wert $FFFF (65535)
A6C9   A0 01      LDY #$01     Zeiger setzen
A6CB   84 0F      STY $0F      und Quote-Modus abschalten
A6CD   B1 5F      LDA ($5F),Y  Linkadresse High holen
A6CF   F0 43      BEQ $A714    Ja: dann fertig
A6D1   20 2C A8   JSR $A82C    prüft auf <Stop>-Taste
A6D4   20 D7 AA   JSR $AAD7    'CR' ausgeben, neue Zeile
A6D7   C8         INY          Zeiger erhöhen
A6D8   B1 5F      LDA ($5F),Y  Zeilenadresse holen (Low)
A6DA   AA         TAX          und in das X-Reg. schieben
A6DB   C8         INY          Zeiger erhöhen
A6DC   B1 5F      LDA ($5F),Y  Zeilenadresse holen (High)
A6DE   C5 15      CMP $15      mit Endnummer vergleichen
A6E0   D0 04      BNE $A6E6    Gleich? Nein: $A6E6
A6E2   E4 14      CPX $14      Low-Nummer vergleichen
A6E4   F0 02      BEQ $A6E8    Gleich? Ja: $A6E8
A6E6   B0 2C      BCS $A714    Größer: dann fertig
A6E8   84 49      STY $49      Y-Reg. zwischenspeichern
A6EA   20 CD BD   JSR $BDCD    Zeilennnummer ausgeben
A6ED   A9 20      LDA #$20     ' ' Leerzeichen
A6EF   A4 49      LDY $49      Y-Reg. wiederholen
A6F1   29 7F      AND #$7F     Bit 7 löschen
A6F3   20 47 AB   JSR $AB47    Zeichen ausgeben
A6F6   C9 22      CMP #$22     '"' Hochkomma ?
A6F8   D0 06      BNE $A700    Nein: $A700
A6FA   A5 0F      LDA $0F      Hochkomma-Flag laden,
A6FC   49 FF      EOR #$FF     umdrehen (NOT)
A6FE   85 0F      STA $0F      und wieder abspeichern
A700   C8         INY          Zeilenende nach 255 Zeichen ?
A701   F0 11      BEQ $A714    Nein: dann aufhören
A703   B1 5F      LDA ($5F),Y  Zeichen holen
A705   D0 10      BNE $A717    kein Zeilenende, dann listen
A707   A8         TAY          Akku als Zeiger nach Y
A708   B1 5F      LDA ($5F),Y  Startadresse der nächsten
A70A   AA         TAX          Zeile holen (Low) und nach X
A70B   C8         INY          Zeiger erhöhen
A70C   B1 5F      LDA ($5F),Y  Adresse der Zeile (High)
A70E   86 5F      STX $5F      als Zeiger merken
A710   85 60      STA $60      (speichern nach $5F/60) und
A712   D0 B5      BNE $A6C9    weitermachen
A714   4C 86 E3   JMP $E386    zum BASIC-Warmstart
```

```
****************************** BASIC-Code in Klartext
                               umwandlen
A717   6C 06 03   JMP ($0306) JMP $A71A

Einsprung von $A717

A71A   10 D7      BPL A6F3    kein Interpretercode:ausgeben
A71C   C9 FF      CMP #$FF    Code für Pi?
A71E   F0 D3      BEQ A6F3    Ja: so ausgeben
A720   24 0F      BIT   0F    Hochkommamodus?
A722   30 CF      BMI A6F3    dann Zeichen so ausgeben
A724   38         SEC         Carry setzen (Subtraktion)
A725   E9 7F      SBC #$7F    Offset abziehen
A727   AA         TAX         Code nach X
A728   84 49      STY $49     Zeichenzeiger merken
A72A   A0 FF      LDY #$FF    Zeiger auf Befehlstabelle
A72C   CA         DEX         erstes Befehlswort?
A72D   F0 08      BEQ $A737   Ja: ausgeben
A72F   C8         INY         Zeiger erhöhen
A730   B9 9E A0   LDA $A09E,Y Offset für X-tes Befehlswort
A733   10 FA      BPL $A72F   alle Zeichen bis zum letzten
A735   30 F5      BMI $A72C   überlesen (Bit 7 gesetzt)
A737   C8         INY         Zeiger erhöhen
A738   B9 9E A0   LDA $A09E,Y Befehlswort aus Tabelle holen
A73B   30 B2      BMI $A6EF   letzter Buchstabe: fertig
A73D   20 47 AB   JSR $AB47   Zeichen ausgeben
A740   D0 F5      BNE $A737   nächsten Buchstaben ausgeben

****************************** BASIC-Befehl  FOR
A742   A9 80      LDA #$80    Wert laden und
A744   85 10      STA $10     Integer sperren
A746   20 A5 A9   JSR $A9A5   LET, setzt FOR-Variable
A749   20 8A A3   JSR $A38A   sucht offene FOR-NEXT-Schlei.
A74C   D0 05      BNE $A753   nicht gefunden: $A753
A74E   8A         TXA         X-Reg. nach Akku
A74F   69 0F      ADC #$0F    Stapelzeiger erhöhen
A751   AA         TAX         Akku zurück nach X-Reg. und
A752   9A         TXS         in den Stapelzeiger
A753   68         PLA         Rücksprungadresse vom Stapel
A754   68         PLA         holen (Low und High)
A755   A9 09      LDA #$09    Wert für Prüfung laden
A757   20 FB A3   JSR $A3FB   prüft auf Platz im Stapel
A75A   20 06 A9   JSR $A906   sucht nächstes BAS.-Statement
A75D   18         CLC         Carry löschen (Addition)
A75E   98         TYA         CHRGET-Zeiger und Offset
A75F   65 7A      ADC $7A     = Startadresse der Schleife
A761   48         PHA         auf Stapel speichern
A762   A5 7B      LDA $7B     High-Byte holen und
A764   69 00      ADC #$00    Übertrag addieren und
A766   48         PHA         auf den Stapel legen
A767   A5 3A      LDA $3A     Aktuelle
A769   48         PHA         Zeilennummer laden und auf
A76A   A5 39      LDA $39     den Stapel schieben
```

```
A76C    48          PHA           (Low und High-Byte)
A76D    A9 A4       LDA #$A4        'TO' - Code
A76F    20 FF AE    JSR $AEFF     prüft auf Code
A772    20 8D AD    JSR $AD8D     prüft ob numerische Variable
A775    20 8A AD    JSR $AD8A     numerischer Ausdruck nach FAC
A778    A5 66       LDA $66       Vorzeichen-Byte von FAC holen
A77A    09 7F       ORA #$7F      Bit 0 bis 6 setzen
A77C    25 62       AND $62       mit $62 angleichen
A77E    85 62       STA $62       und abspeichern
A780    A9 8B       LDA #$8B      Rücksprungadresse laden
A782    A0 A7       LDY #$A7      (Low und High)
A784    85 22       STA $22       und zwischenspeichern
A786    84 23       STY $23       (Low und High)
A788    4C 43 AE    JMP $AE43     Schleifenendwert auf Stapel
A78B    A9 BC       LDA #$BC      Zeiger auf Konstante 1 setzen
A78D    A0 B9       LDY #$B9      (Ersatzwert für STEP)
A78F    20 A2 BB    JSR $BBA2     als Default-STEP-Wert in FAC
A792    20 79 00    JSR $0079     CHRGOT: letztes Zeichen holen
A795    C9 A9       CMP #$A9      'STEP' - Code?
A797    D0 06       BNE $A79F     kein STEP-Wert: $A79F
A799    20 73 00    JSR $0073     CHRGET nächstes Zeichen holen
A79C    20 8A AD    JSR $AD8A     numerischer Ausdruck nach FAC
A79F    20 2B BC    JSR $BC2B     holt Vorzeichen-Byte
A7A2    20 38 AE    JSR $AE38     Vorz. und STEP-Wert auf Stack
A7A5    A5 4A       LDA $4A       Zeiger auf Variablenwert
A7A7    48          PHA           (Low) auf den Stapel
A7A8    A5 49       LDA $49       Zeiger (High)
A7AA    48          PHA           auf den Stapel
A7AB    A9 81       LDA #$81      und FOR-Code
A7AD    48          PHA           auf den Stapel legen

****************************      Interpreterschleife

Einsprung von $A7EA, $A89D, $AD75

A7AE    20 2C A8    JSR $A82C     prüft auf <Stop>-Taste
A7B1    A5 7A       LDA $7A       CHRGET Zeiger (Low und High)
A7B3    A4 7B       LDY $7B       laden
A7B5    C0 02       CPY #$02      Direktmodus?
A7B7    EA          NOP           No OPeration
A7B8    F0 04       BEQ $A7BE     ja: $A7BE
A7BA    85 3D       STA $3D       als Zeiger für CONT
A7BC    84 3E       STY $3E       merken
A7BE    A0 00       LDY #$00      Zeiger setzen
A7C0    B1 7A       LDA ($7A),Y   laufendes Zeichen holen
A7C2    D0 43       BNE $A807     nicht Zeilenende?
A7C4    A0 02       LDY #$02      Zeiger neu setzen
A7C6    B1 7A       LDA ($7A),Y   Programmende?
A7C8    18          CLC           Flag für END setzen
A7C9    D0 03       BNE $A7CE     Kein Programmende: $A7CE
A7CB    4C 4B A8    JMP $A84B     ja: dann END ausführen
A7CE    C8          INY           Zeiger erhöhen
A7CF    B1 7A       LDA ($7A),Y   laufende Zeilennummer
```

```
A7D1   85 39      STA $39       (Low) nach $39
A7D3   C8         INY           Zeiger auf nächstes Byte
A7D4   B1 7A      LDA ($7A),Y   laufende Zeilennummer
A7D6   85 3A      STA $3A       (High) nach $3A
A7D8   98         TYA           Zeiger nach Akku
A7D9   65 7A      ADC $7A       Programmzeiger auf
A7DB   85 7A      STA $7A       Programmzeile setzen
A7DD   90 02      BCC $A7E1     C=0: Erhöhung umgehen
A7DF   E6 7B      INC $7B       Programmzeiger (High) erhöhen

Einsprung von $A499

A7E1   6C 08 03   JMP ($0308)   JMP $A7E4 Statement ausführen

Einsprung von $A7E1

A7E4   20 73 00   JSR $0073     CHRGET nächstes Zeichen holen
A7E7   20 ED A7   JSR $A7ED     Statement ausführen
A7EA   4C AE A7   JMP $A7AE     zurück zur Interpreterschlei.

****************************    BASIC-Statement ausführen

Einsprung von $A7E7, $A948

A7ED   F0 3C      BEQ $A82B     Zeilenende, dann fertig

Einsprung von $A95C

A7EF   E9 80      SBC #$80      Token?
A7F1   90 11      BCC $A804     nein: dann zum LET-Befehl
A7F3   C9 23      CMP #$23      NEW?
A7F5   B0 17      BCS $A80E     Funktions-Token oder GO TO
A7F7   0A         ASL           BASIC-Befehl, Code mal 2
A7F8   A8         TAY           als Zeiger ins Y-Reg.
A7F9   B9 0D A0   LDA $A00D,Y   Befehlsadresse (Low und
A7FC   48         PHA           High) aus Tabelle
A7FD   B9 0C A0   LDA $A00C,Y   holen und als
A800   48         PHA           Rücksprungadresse auf Stapel
A801   4C 73 00   JMP $0073     Zeichen und Befehl ausführen
A804   4C A5 A9   JMP $A9A5     zum LET-Befehl

****************************
A807   C9 3A      CMP #$3A      ':' ist es Doppelpunkt?
A809   F0 D6      BEQ $A7E1     ja: $A7E1
A80B   4C 08 AF   JMP $AF08     sonst 'SYNTAX ERROR'

****************************    prüft auf 'GO' 'TO' Code
A80E   C9 4B      CMP #$4B      'GO'  (minus $80)
A810   D0 F9      BNE $A80B     nein: 'SYNTAX ERROR'
A812   20 73 00   JSR $0073     nächstes Zeichen holen
A815   A9 A4      LDA #$A4      'TO'
```

```
A817   20 FF AE   JSR $AEFF   prüft auf Code
A81A   4C A0 A8   JMP $A8A0   zum GOTO-Befehl

****************************  BASIC-Befehl  RESTORE

Einsprung von $A677

A81D   38         SEC         Carry setzen (Subtraktion)
A81E   A5 2B      LDA $2B     Programmstartzeiger (Low)
A820   E9 01      SBC #$01    laden und davon 1 abziehen
A822   A4 2C      LDY $2C     und High-Byte holen
A824   B0 01      BCS $A827
A826   88         DEY         Low-Byte -1

Einsprung von $ACE7

A827   85 41      STA $41     als DATA-Zeiger
A829   84 42      STY $42     abspeichern
A82B   60         RTS         Rücksprung

****************************  prüft auf <Stop>-Taste

Einsprung von $A6D1, $A7AE

A82C   20 E1 FF   JSR $FFE1   <Stop>-Taste abfragen

****************************  BASIC-Befehl  STOP
A82F   B0 01      BCS $A832   C=1: Flag für STOP

****************************  BASIC-Befehl  END
A831   18         CLC         C=0  Flag für END
A832   D0 3C      BNE $A870   <Run/Stop> nicht gedrückt: RTS
A834   A5 7A      LDA $7A     Programmzeiger laden
A836   A4 7B      LDY $7B     (Low und High-Byte)
A838   A6 3A      LDX $3A     Direktmodus?
A83A   E8         INX         (Zeilennummer -1)
A83B   F0 0C      BEQ $A849   ja: $A849
A83D   85 3D      STA $3D     als Zeiger für CONT setzen
A83F   84 3E      STY $3E     (Low und High)
A841   A5 39      LDA $39     Nummer der laufenden Zeile
A843   A4 3A      LDY $3A     holen (Low und High)
A845   85 3B      STA $3B     und als Zeilennummer für
A847   84 3C      STY $3C     CONT  merken
A849   68         PLA         Rücksprungadresse
A84A   68         PLA         vom Stapel entfernen

Einsprung von $A7CB

A84B   A9 81      LDA #$81    Zeiger auf Startadresse
A84D   A0 A3      LDY #$A3    BREAK setzen
A84F   90 03      BCC $A854   END Flag?
```

```
A851    4C 69 A4    JMP $A469    nein: 'BREAK IN XXX' ausgeben
A854    4C 86 E3    JMP $E386    zum BASIC-Warmstart

****************************     BASIC-Befehl  CONT
A857    D0 17       BNE $A870    Kein Trennzeichen: SYNTAX ERR
A859    A2 1A       LDX #$1A     Fehlernr. für 'CAN'T CONTINUE
A85B    A4 3E       LDY $3E      CONT gesperrt?
A85D    D0 03       BNE $A862    nein: $A862
A85F    4C 37 A4    JMP $A437    Fehlermeldung ausgeben
A862    A5 3D       LDA $3D      CONT-Zeiger (Low) laden
A864    85 7A       STA $7A      und CONT-Zeiger als Programm-
A866    84 7B       STY $7B      zeiger abspeichern
A868    A5 3B       LDA $3B      und
A86A    A4 3C       LDY $3C      Zeilennummer wieder
A86C    85 39       STA $39      setzen
A86E    84 3A       STY $3A      (Low- und High-Byte)
A870    60          RTS          Rücksprung

****************************     BASIC-Befehl  RUN
A871    08          PHP          Status-Register retten
A872    A9 00       LDA #$00     Wert laden und
A874    20 90 FF    JSR $FF90    Flag für Programmodus setzen
A877    28          PLP          Status-Register zurückholen
A878    D0 03       BNE $A87D    weitere Zeichen (Zeilennr.)?
A87A    4C 59 A6    JMP $A659    Programmzeiger setzen, CLR

A87D    20 60 A6    JSR $A660    CLR-Befehl
A880    4C 97 A8    JMP $A897    GOTO-Befehl

****************************     BASIC-Befehl  GOSUB
A883    A9 03       LDA #$03     Wert für Prüfung
A885    20 FB A3    JSR $A3FB    prüft auf Platz im Stapel
A888    A5 7B       LDA $7B      Programmzeiger (Low-
A88A    48          PHA          und High-Byte) laden
A88B    A5 7A       LDA $7A      und auf den
A88D    48          PHA          Stapel retten
A88E    A5 3A       LDA $3A      Zeilennummer laden (High)
A890    48          PHA          und auf den Stapel legen
A891    A5 39       LDA $39      Zeilennummer Low laden
A893    48          PHA          und auf den Stapel legen
A894    A9 8D       LDA #$8D     'GOSUB'-Code laden
A896    48          PHA          und auf den Stapel legen

Einsprung von $A880

A897    20 79 00    JSR $0079    CHRGOT: letztes Zeichen holen
A89A    20 A0 A8    JSR $A8A0    GOTO-Befehl
A89D    4C AE A7    JMP $A7AE    zur Interpreterschleife

****************************     BASIC-Befehl  GOTO

Einsprung von $A81A, $A89A, $A945
```

```
A8A0   20 6B A9   JSR $A96B    Zeilennummer nach $14/$15
A8A3   20 09 A9   JSR $A909    nächsten Zeilenanfang suchen
A8A6   38         SEC          Carry löschen (Subtraktion)
A8A7   A5 39      LDA $39      aktuelle Zeilennummer (Low)
A8A9   E5 14      SBC $14      kleiner als laufende Zeile?
A8AB   A5 3A      LDA $3A      aktuelle Zeilennummer (High)
A8AD   E5 15      SBC $15      kleiner als laufende Zeile?
A8AF   B0 0B      BCS $A8BC    nein: $A8BC
A8B1   98         TYA          Differenz in Akku
A8B2   38         SEC          Carry setzen (Addition)
A8B3   65 7A      ADC $7A      Programmzeiger addieren
A8B5   A6 7B      LDX $7B      sucht ab laufender Zeile
A8B7   90 07      BCC $A8C0    unbedingter
A8B9   E8         INX          Sprung
A8BA   B0 04      BCS $A8C0    zu $A8C0
A8BC   A5 2B      LDA $2B      sucht ab Programmstart
A8BE   A6 2C      LDX $2C
A8C0   20 17 A6   JSR $A617    sucht Programmzeile
A8C3   90 1E      BCC $A8E3    nicht gefunden: 'undef'd st.'
A8C5   A5 5F      LDA $5F      von der Startadresse (Zeile)
A8C7   E9 01      SBC #$01     eins subtrahieren und als
A8C9   85 7A      STA $7A      Programmzeiger (Low)
A8CB   A5 60      LDA $60      High-Byte der Zeile laden
A8CD   E9 00      SBC #$00     Übertrag berücksichtigen
A8CF   85 7B      STA $7B      und als Programmzeiger
A8D1   60         RTS          Rücksprung

*****************************  BASIC-Befehl  RETURN
A8D2   D0 FD      BNE $A8D1    Kein Trennzeichen: SYNTAX ERR
A8D4   A9 FF      LDA #$FF     Wert laden und
A8D6   85 4A      STA $4A      FOR-NEXT-ZEIGER neu setzen
A8D8   20 8A A3   JSR $A38A    GOSUB-Datensatz suchen
A8DB   9A         TXS
A8DC   C9 8D      CMP #$8D     'GOSUB'-Code?
A8DE   F0 0B      BEQ $A8EB    ja: $A8E8
A8E0   A2 0C      LDX #$0C     Nr für 'return without gosub'
A8E2   2C         .BYTE $2C
A8E3   A2 11      LDX #$02     Nr für 'undef'd statement'
A8E5   4C 37 A4   JMP $A437    Fehlermeldung ausgeben
A8E8   4C 08 AF   JMP $AF08    'syntax error' ausgeben

A8EB   68         PLA          GOSUB-Code vom Stapel holen
A8EC   68         PLA          Zeilennummer (Low) wieder-
A8ED   85 39      STA $39      holen und abspeichern
A8EF   68         PLA          Zeilennummer (High) holen
A8F0   85 3A      STA $3A      und abspeichern
A8F2   68         PLA          Programmzeiger (Low) wieder-
A8F3   85 7A      STA $7A      holen und abspeichern
A8F5   68         PLA          Programmzeiger (High) holen
A8F6   85 7B      STA $7B      abspeichern

*****************************  BASIC-Befehl  DATA
```

```
Einsprung von $ABE7, $B3DB

A8F8    20 06 A9    JSR $A906     nächstes Statement suchen

Einsprung von $ABF6, $ACD1

A8FB    98          TYA           Offset
A8FC    18          CLC           Carry löschen (Addition)
A8FD    65 7A       ADC $7A       Programmzeiger addieren
A8FF    85 7A       STA $7A       und wieder abspeichern
A901    90 02       BCC $A905     Verminderung übergehen
A903    E6 7B       INC $7B       Programmzeiger vermindern
A905    60          RTS           Rücksprung

****************************      Offset des nächsten
                                  Trennzeichens finden

Einsprung von $A75A, $A8F8, $ABF3, $ACB8

A906    A2 3A       LDX #$3A      ':' Doppelpunkt
A908    2C          .BYTE $2C

Einsprung von $A8A3, $A93B

A909    A2 00       LDX #$00      $0  Zeilenende
A90B    86 07       STX $07       als Suchzeichen
A90D    A0 00       LDY #$00      Zähler
A90F    84 08       STY $08       initialisieren
A911    A5 08       LDA $08       Speicherzelle $7
A913    A6 07       LDX $07       gesuchtes Zeichen
A915    85 07       STA $07       mit $8
A917    86 08       STX $08       vertauschen
A919    B1 7A       LDA ($7A),Y   Zeichen holen
A91B    F0 E8       BEQ $A905     Zeilenende, dann fertig
A91D    C5 08       CMP $08       = Suchzeichen?
A91F    F0 E4       BEQ $A905     ja: $A905
A921    C8          INY           Zeiger erhöhen
A922    C9 22       CMP #$22      '"' Hochkomma?
A924    D0 F3       BNE $A919     nein: $A919
A926    F0 E9       BEQ $A911     sonst $7 und $8 vertauschen

****************************      BASIC-Befehl  IF
A928    20 9E AD    JSR $AD9E     FRMEVL  Ausdruck berechnen
A92B    20 79 00    JSR $0079     CHRGOT  letztes Zeichen
A92E    C9 89       CMP #$89      'GOTO'-Code?
A930    F0 05       BEQ $A937     ja: $A937
A932    A9 A7       LDA #$A7      'THEN'-Code
A934    20 FF AE    JSR $AEFF     prüft auf Code
A937    A5 61       LDA $61       Ergebnis des IF-Ausdrucks
A939    D0 05       BNE $A940     Ausdruck wahr?

****************************      BASIC-Befehl  REM
```

```
A93B   20 09 A9   JSR $A909   Nein: Zeilenanfang suchen
A93E   F0 BB      BEQ $A8FB   Programmz. auf nächste Zeile

A940   20 79 00   JSR $0079   CHRGOT: letztes Zeichen holen
A943   B0 03      BCS $A948   keine Ziffer?
A945   4C A0 A8   JMP $A8A0   zum GOTO-Befehl
A948   4C ED A7   JMP $A7ED   Befehl decodieren, ausführen

****************************  BASIC-Befehl  ON
A94B   20 9E B7   JSR $B79E   Byte-Wert (0 bis 255) holen
A94E   48         PHA         Code merken
A94F   C9 8D      CMP #$8D    'GOSUB'-Code?
A951   F0 04      BEQ $A957   Ja: $A957
A953   C9 89      CMP #$89    'GOTO'-Code?
A955   D0 91      BNE $A8E8   Nein: dann 'SYNTAX ERROR'
A957   C6 65      DEC $65     Zähler vermindern
A959   D0 04      BNE $A95F   Noch nicht null?
A95B   68         PLA         Ja: Code zurückholen
A95C   4C EF A7   JMP $A7EF   und Befehl ausführen

A95F   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
A962   20 6B A9   JSR $A96B   Zeilennummer holen
A965   C9 2C      CMP #$2C    ',' Komma?
A967   F0 EE      BEQ $A957   Ja: dann weiter
A969   68         PLA         Kein Sprung: Code zurückholen
A96A   60         RTS         Rücksprung

****************************  Zeilennummer nach $14/$15

Einsprung von $A49C, $A6A4, $A6B6, A8A0, $A962

A96B   A2 00      LDX #$00    Wert laden und
A96D   86 14      STX $14     Vorsetzen
A96F   86 15      STX $15     (für Zeilennummer gleich 0)

Einsprung von $A9A2

A971   B0 F7      BCS $A96A   Keine Ziffer, dann fertig
A973   E9 2F      SBC #$2F    '0'-1 abziehen, gibt Hexwert
A975   85 07      STA $07     merken
A977   A5 15      LDA $15     High-Byte holen
A979   85 22      STA $22     zwischenspeichern
A97B   C9 19      CMP #$19    Zahl bereits größer 6400?
A97D   B0 D4      BCS $A953   dann 'SYNTAX ERROR'
A97F   A5 14      LDA $14     Zahl * 10 (= *2*2+Zahl*2)
A981   0A         ASL         Wert und Zwischenwert je
A982   26 22      ROL $22     2 mal um 1 Bit nach
A984   0A         ASL         links rollen
A985   26 22      ROL $22     (entspricht 2 * 2)
A987   65 14      ADC $14     plus ursprünglicher Wert
A989   85 14      STA $14     und abspeichern
A98B   A5 22      LDA $22     Zwischenwert zu
A98D   65 15      ADC $15     zweitem Wert addieren
```

```
A98F   85 15      STA $15      und wieder abspeichern
A991   06 14      ASL $14      Speicherzelle $14 und
A993   26 15      ROL $15      $15 verdoppeln
A995   A5 14      LDA $14      Wert wieder laden
A997   65 07      ADC $07      und Einerziffer addieren
A999   85 14      STA $14      wieder speichern
A99B   90 02      BCC $A99F    Carry gesetzt? (Übertrag)
A99D   E6 15      INC $15      Übertrag addieren
A99F   20 73 00   JSR $0073    CHRGET nächstes Zeichen holen
A9A2   4C 71 A9   JMP $A971    und weiter machen

******************************  BASIC-Befehl  LET

Einsprung von $A746, $A804

A9A5   20 8B B0   JSR $B08B    sucht Variable hinter LET
A9A8   85 49      STA $49      und Variablenadresse
A9AA   84 4A      STY $4A      merken (Low- und High-Byte)
A9AC   A9 B2      LDA #$B2     '=' - Code
A9AE   20 FF AE   JSR $AEFF    prüft auf Code
A9B1   A5 0E      LDA $0E      Integer-Flag
A9B3   48         PHA          auf Stapel retten
A9B4   A5 0D      LDA $0D      und Typ-Flag
A9B6   48         PHA          (String/numerisch) retten
A9B7   20 9E AD   JSR $AD9E    FRMEVL: Ausdruck holen
A9BA   68         PLA          Typ-Flag wiederholen
A9BB   2A         ROL  A       und Bit 7 ins Carry schieben
A9BC   20 90 AD   JSR $AD90    auf richtigen Typ prüfen
A9BF   D0 18      BNE $A9D9    String? ja: $A9D9
A9C1   68         PLA          Integer-Flag zurückholen

Einsprung von $AC8E

A9C2   10 12      BPL $A9D6    INTEGER? ja: $A9D6

******************************  Wertzuweisung INTEGER
A9C4   20 1B BC   JSR $BC1B    FAC runden
A9C7   20 BF B1   JSR $B1BF    und nach INTEGER wandeln
A9CA   A0 00      LDY #$00     Zeiger setzen
A9CC   A5 64      LDA $64      High-Byte holen und
A9CE   91 49      STA ($49),Y  Wert in Variable bringen
A9D0   C8         INY          Zeiger erhöhen
A9D1   A5 65      LDA $65      Low-Byte holen und
A9D3   91 49      STA ($49),Y  Wert in Variable bringen
A9D5   60         RTS          Rücksprung

******************************  Wertzuweisung REAL
A9D6   4C D0 BB   JMP $BBD0    FAC nach Variable bringen

******************************  Wertzuweisung String
A9D9   68         PLA          Akku vom Stapel holen
```

Einsprung von $AC83

```
A9DA   A4 4A      LDY $4A      Variablenadresse (High) holen
A9DC   C0 BF      CPY #$BF     Ist Variable TI$?
A9DE   D0 4C      BNE $AA2C    Nein: $AA2C
A9E0   20 A6 B6   JSR $B6A6    FRESTR
A9E3   C9 06      CMP #$06     String-Länge gleich 6
A9E5   D0 3D      BNE $AA24    nein: 'illegal quantity'
A9E7   A0 00      LDY #$00     Wert holen
A9E9   84 61      STY $61      und damit FAC
A9EB   84 66      STY $66      initialisieren
A9ED   84 71      STY $71      (Akku, Vorzeichen und Zeiger)
A9EF   20 1D AA   JSR $AA1D    prüft nächstes Z. auf Ziffer
A9F2   20 E2 BA   JSR $BAE2    FAC = FAC * 10
A9F5   E6 71      INC $71      Stellenzähler erhöhen
A9F7   A4 71      LDY $71      und ins Y-Reg. bringen
A9F9   20 1D AA   JSR $AA1D    prüft nächstes Z. auf Ziffer
A9FC   20 0C BC   JSR $BC0C    FAC nach ARG kopieren
A9FF   AA         TAX          FAC gleich 0?
AA00   F0 05      BEQ $AA07    Ja: $AA07
AA02   E8         INX          Exponent von FAC erhöhen
AA03   8A         TXA          (FAC *2) und in den Akku
AA04   20 ED BA   JSR $BAED    FAC = FAC + ARG
AA07   A4 71      LDY $71      Stellenzähler
AA09   C8         INY          erhöhen
AA0A   C0 06      CPY #$06     schon 6 Stellen?
AA0C   D0 DF      BNE $A9ED    Nein: nächstes Zeichen
AA0E   20 E2 BA   JSR $BAE2    FAC = FAC * 10
AA11   20 9B BC   JSR $BC9B    FAC rechtsbündig machen
AA14   A6 64      LDX $64      Werte für
AA16   A4 63      LDY $63      eingegebene Uhrzeit
AA18   A5 65      LDA $65      holen und
AA1A   4C DB FF   JMP $FFDB    Time setzen
```

```
****************************** Zeichen auf Ziffer prüfen
```

Einsprung von $A9EF, $A9F9

```
AA1D   B1 22      LDA ($22),Y Zeichen holen (aus String)
AA1F   20 80 00   JSR $0080    auf Ziffer prüfen
AA22   90 03      BCC $AA27    Ziffer: $AA27
AA24   4C 48 B2   JMP $B248    sonst: 'illegal quantity'
AA27   E9 2F      SBC #$2F     von ASCII nach HEX umwandeln
AA29   4C 7E BD   JMP $BD7E    in FAC und ARG übertragen
```

```
****************************** Wertzuweisung an normalen
                               String
AA2C   A0 02      LDY #$02     Zeiger setzen
AA2E   B1 64      LDA ($64),Y String-Adresse High mit
AA30   C5 34      CMP $34      String-Anfangsadr. vergleichen
AA32   90 17      BCC $AA4B    kleiner: String im Programm
AA34   D0 07      BNE $AA3D    größer: $AA3D
AA36   88         DEY          Zeiger vermindern
```

```
AA37   B1 64      LDA ($64),Y String-Adresse (Low) holen
AA39   C5 33      CMP $33     und vergleichen
AA3B   90 0E      BCC $AA4B   kleiner: String im Programm
AA3D   A4 65      LDY $65     Zeiger auf String-Descriptor
AA3F   C4 2E      CPY $2E     mit Variablenstart vergl.
AA41   90 08      BCC $AA4B   kleiner: $AA4B
AA43   D0 0D      BNE $AA52   größer: $AA52
AA45   A5 64      LDA $64     String-Descriptorzeiger (Low)
AA47   C5 2D      CMP $2D     mit Variablenstart vergl.
AA49   B0 07      BCS $AA52   größer: $AA52
AA4B   A5 64      LDA $64     Zeiger in Akku und Y-Reg.
AA4D   A4 65      LDY $65     auf String-Descriptor setzen
AA4F   4C 68 AA   JMP $AA68   bis $AA68 überspringen
AA52   A0 00      LDY #$00    Zeiger setzen
AA54   B1 64      LDA ($64),Y Länge des Strings holen
AA56   20 75 B4   JSR $B475   prüft Platz, setzt Stringz.
AA59   A5 50      LDA $50     Zeiger auf String-Descriptor
AA5B   A4 51      LDY $51     holen (Low- und High-Byte)
AA5D   85 6F      STA $6F     und
AA5F   84 70      STY $70     speichern
AA61   20 7A B6   JSR $B67A   String in Bereich übertragen
AA64   A9 61      LDA #$61    Werte laden
AA66   A0 00      LDY #$00    und damit

Einsprung von $AAeF

AA68   85 50      STA $50     String-Descriptor
AA6A   84 51      STY $51     neu setzen
AA6C   20 DB B6   JSR $B6DB   Descriptor löschen
AA6F   A0 00      LDY #$00    Zeiger setzen
AA71   B1 50      LDA ($50),Y Länge des Descriptors holen
AA73   91 49      STA ($49),Y und abspeichern
AA75   C8         INY         Zeiger erhöhen
AA76   B1 50      LDA ($50),Y Adresse (Low) holen
AA78   91 49      STA ($49),Y und speichern
AA7A   C8         INY         Zeiger erhöhen
AA7B   B1 50      LDA ($50),Y und Adresse (High)
AA7D   91 49      STA ($49),Y in Variable bringen
AA7F   60         RTS         Rücksprung

***************************** BASIC-Befehl  PRINT#
AA80   20 86 AA   JSR $AA86   CMD-Befehl
AA83   4C B5 AB   JMP $ABB5   und CLRCH

***************************** BASIC-Befehl  CMD

Einsprung von $AA80

AA86   20 9E B7   JSR $B79E   holt Byte-Ausdruck
AA89   F0 05      BEQ $AA90   Trennzeichen: $AA90
AA8B   A9 2C      LDA #$2C    ',', Wert laden
AA8D   20 FF AE   JSR $AEFF   prüft auf Komma
AA90   08         PHP         Status-Register merken
```

```
AA91   86 13      STX $13      Nr. des Ausgabegeräts merken
AA93   20 18 E1   JSR $E118    CKOUT, Ausgabegerät setzen
AA96   28         PLP          Status-Register wiederholen
AA97   4C A0 AA   JMP $AAA0    zum PRINT-Befehl
AA9A   20 21 AB   JSR $AB21    String drucken
AA9D   20 79 00   JSR $0079    CHRGOT  letztes Zeichen

******************************  BASIC-Befehl  PRINT

Einsprung von $AA97

AAA0   F0 35      BEQ $AAD7    Trennzeichen: $AAD7

Einsprung von $AB16

AAA2   F0 43      BEQ $AAE7    Trennz. (TAB, SPC): RTS
AAA4   C9 A3      CMP #$A3     'TAB('-Code?
AAA6   F0 50      BEQ $AAF8    Ja: $AAF8
AAA8   C9 A6      CMP #$A6     'SPC('-Code?
AAAA   18         CLC          Flag für SPC setzen
AAAB   F0 4B      BEQ $AAF8    SPC-Code: $AAF8
AAAD   C9 2C      CMP #$2C     ','-Code? (Komma)
AAAF   F0 37      BEQ $AAE8    Ja: $AAE8
AAB1   C9 3B      CMP #$3B     ';'-Code? (Semikolon)
AAB3   F0 5E      BEQ $AB13    Ja: nächstes Zeichen, weiter
AAB5   20 9E AD   JSR $AD9E    FRMEVL: Term holen
AAB8   24 0D      BIT $0D      Typ-Flag
AABA   30 DE      BMI $AA9A    String?
AABC   20 DD BD   JSR $BDDD    FAC in ASCII-String wandeln
AABF   20 87 B4   JSR $B487    String-Parameter holen
AAC2   20 21 AB   JSR $AB21    String drucken
AAC5   20 3B AB   JSR $AB3B    Cursor right bzw. Leerzeichen
AAC8   D0 D3      BNE $AA9D    weiter machen

Einsprung von $A576

AACA   A9 00      LDA #$00     Eingabepuffer
AACC   9D 00 02   STA $0200,X mit $0 abschließen
AACF   A2 FF      LDX #$FF     Zeiger auf
AAD1   A0 01      LDY #$01     Eingabepuffer ab $0200 setzen
AAD3   A5 13      LDA $13      Nummer des Ausgabegeräts
AAD5   D0 10      BNE $AAE7    Tastatur? Nein: RTS

Einsprung von $A44E, $A6D4

AAD7   A9 0D      LDA #$0D     'CR' Carriage Return
AAD9   20 47 AB   JSR $AB47    ausgeben
AADC   24 13      BIT $13      logische File-Nummer
AADE   10 05      BPL $AAE5    kleiner 128?
AAE0   A9 0A      LDA #$0A     'LF' line feed
AAE2   20 47 AB   JSR $AB47    ausgeben
```

```
Einsprung von $AB35

AAE5    49 FF       EOR #$FF    NOT
AAE7    60          RTS         Rücksprung

AAE8    38          SEC         Zehner-Tabulator mit Komma
AAE9    20 F0 FF    JSR $FFF0   Cursor-Position holen
AAEC    98          TYA         Spalte ins Y-Reg.
AAED    38          SEC         Carry setzen (Subtr.)
AAEE    E9 0A       SBC #$0A    10 abziehen
AAF0    B0 FC       BCS $AAEE   nicht negativ?
AAF2    49 FF       EOR #$FF    invertieren
AAF4    69 01       ADC #$01    +1 (Zweierkomplement)
AAF6    D0 16       BNE $AB0E   unbedingter Sprung

****************************    TAB(  (C=1)  und SPC( (C=0)
AAF8    08          PHP         Flags merken
AAF9    38          SEC         Carry setzen
AAFA    20 F0 FF    JSR $FFF0   Cursor-Position holen
AAFD    84 09       STY $09     und Spalte merken
AAFF    20 9B B7    JSR $B79B   Byte-Wert holen
AB02    C9 29       CMP #$29    ')' Klammer zu?
AB04    D0 59       BNE $AB5F   nein: 'SYNTAX ERROR'
AB06    28          PLP         Flags wiederherstellen
AB07    90 06       BCC $AB0F   zu SPC(
AB09    8A          TXA         TAB-Wert in Akku
AB0A    E5 09       SBC $09     mit Cursor-Spalte vergleichen
AB0C    90 05       BCC $AB13   kleiner Cursor-Position: RTS
AB0E    AA          TAX         Schritte bis zum Tabulator
AB0F    E8          INX         aus Zähler initialisieren
AB10    CA          DEX         um 1 vermindern
AB11    D0 06       BNE $AB19   =0? nein: Cursor right
AB13    20 73 00    JSR $0073   nächstes Zeichen holen
AB16    4C A2 AA    JMP $AAA2   und weitermachen

AB19    20 3B AB    JSR $AB3B   Cursor right bzw. Leerzeichen
AB1C    D0 F2       BNE $AB10   zum Schleifenanfang

****************************    String ausgeben

Einsprung von $A469, $A478, $AB6F, $ACF8, $BDDA, $E191
$E42D, $E441, $FB8B

AB1E    20 87 B4    JSR $B487   String-Parameter holen

Einsprung von $AA9A, AAC2, $ABCB

AB21    20 A6 B6    JSR $B6A6   FRESTR
AB24    AA          TAX         String-Länge
AB25    A0 00       LDY #$00    Zeiger für String-Ausgabe
AB27    E8          INX         erhöhen
```

```
Einsprung von $AB38

AB28   CA         DEX            vermindern
AB29   F0 BC      BEQ $AAE7      String zu Ende?
AB2B   B1 22      LDA ($22),Y    Zeichen des Strings
AB2D   20 47 AB   JSR $AB47      ausgeben
AB30   C8         INY            Zeiger erhöhen
AB31   C9 0D      CMP #$0D       'CR' Carriage Return?
AB33   D0 F3      BNE $AB28      nein: weiter
AB35   20 E5 AA   JSR $AAE5      Fehler ! Test auf LF-Ausgabe
AB38   4C 28 AB   JMP $AB28      und weitermachen
*****************************    Ausgabe eines Leerzeichens
                                 bzw. Cursor right

Einsprung von $AAC5, $AB19, $AC00

AB3B   A5 13      LDA $13        Ausgabe in File?
AB3D   F0 03      BEQ $AB42      Bildschirm: dann Cursor right
AB3F   A9 20      LDA #$20       ' ' Leerzeichencode laden
AB41   2C         .BYTE $2C
AB42   A9 1D      LDA #$1D       Cursor-right-Code laden
AB44   2C         .BYTE $2C

Einsprung von $A451, $ABFD, $AC47

AB45   A9 3F      LDA #$3F       '?' Fragezeichencode laden

Einsprung von $A45B, $A6F3, $A73D, $AAD9, $AAE2, $AB2D

AB47   20 0C E1   JSR $E10C      Code ausgeben
AB4A   29 FF      AND #$FF       Flags setzen
AB4C   60         RTS            Rücksprung

*****************************    Fehlerbehandlung bei Eingabe

Einsprung von $AC9A

AB4D   A5 11      LDA $11        Flag für INPUT / GET / READ
AB4F   F0 11      BEQ $AB62      INPUT: $AB62
AB51   30 04      BMI $AB57      READ: $AB57
AB53   A0 FF      LDY #$FF       GET:
AB55   D0 04      BNE $AB5B      unbedingter Sprung

*****************************    Fehler bei READ
AB57   A5 3F      LDA $3F        DATA-Zeilennummer
AB59   A4 40      LDY $40        holen (Low- und High-Byte)

*****************************    Fehler bei GET
AB5B   85 39      STA $39        gleiche Zeilennummer
AB5D   84 3A      STY $3A        des Fehlers
AB5F   4C 08 AF   JMP $AF08      'SYNTAX ERROR'
```

```
******************************  Fehler bei INPUT
AB62   A5 13      LDA $13      Nummer des Eingabegeräts
AB64   F0 05      BEQ $AB6B    Tastatur: 'REDO FROM START'
AB66   A2 18      LDX #$18     Nummer für 'FILE DATA'
AB68   4C 37 A4   JMP $A437    Fehlermeldung ausgeben
AB6B   A9 0C      LDA #$0C     Zeiger in Akku und Y-Reg.
AB6D   A0 AD      LDY #$AD     auf '?REDO FROM START'
AB6F   20 1E AB   JSR $AB1E    String ausgeben
AB72   A5 3D      LDA $3D      Werte holen und
AB74   A4 3E      LDY $3E      Programmzeiger
AB76   85 7A      STA $7A      zurücksetzen
AB78   84 7B      STY $7B      auf INPUT-Befehl
AB7A   60         RTS          Rücksprung

******************************  BASIC-Befehl  GET
AB7B   20 A6 B3   JSR $B3A6    Testet auf Direktmodus
AB7E   C9 23      CMP #$23     folgt '#'?
AB80   D0 10      BNE $AB92    nein: $AB92
AB82   20 73 00   JSR $0073    CHRGET nächstes Zeichen holen
AB85   20 9E B7   JSR $B79E    Byte-Wert holen
AB88   A9 2C      LDA #$2C     ',' Komma
AB8A   20 FF AE   JSR $AEFF    prüft auf Code
AB8D   86 13      STX $13      File-Nummer
AB8F   20 1E E1   JSR $E11E    CHKIN, Eingabe vorbereiten
AB92   A2 01      LDX #$01     Zeiger auf
AB94   A0 02      LDY #$02     Pufferende = $201 ein Zeichen
AB96   A9 00      LDA #$00     Wert laden und
AB98   8D 01 02   STA $0201    Puffer mit $0 abschließen
AB9B   A9 40      LDA #$40     GET-Flag
AB9D   20 0F AC   JSR $AC0F    Wertzuweisung an Variable
ABA0   A6 13      LDX $13      Eingabegerät
ABA2   D0 13      BNE $ABB7    nicht Tastatur, dann CLRCH
ABA4   60         RTS          Rücksprung

******************************  BASIC-Befehl  INPUT#
ABA5   20 9E B7   JSR $B79E    holt Byte-Wert
ABA8   A9 2C      LDA #$2C     ',' Code für Komma
ABAA   20 FF AE   JSR $AEFF    prüft auf Komma
ABAD   86 13      STX $13      Eingabegerät
ABAF   20 1E E1   JSR $E11E    CHKIN, Eingabe vorbereiten
ABB2   20 CE AB   JSR $ABCE    INPUT ohne Dialog-String

Einsprung von $AA83, $ABE4

ABB5   A5 13      LDA $13      Eingabegerät im Akku
ABB7   20 CC FF   JSR $FFCC    setzt Eingabegerät zurück
ABBA   A2 00      LDX #$00     Wert laden und
ABBC   86 13      STX $13      Eingabegerät wieder Tastatur
ABBE   60         RTS          Rücksprung

******************************  BASIC-Befehl  INPUT
ABBF   C9 22      CMP #$22     '"' Hochkomma?
ABC1   D0 0B      BNE $ABCE    nein: $ABDE
```

```
ABC3   20 BD AE   JSR $AEBD    Dialog-String holen
ABC6   A9 3B      LDA #$3B     ';' Semikolon
ABC8   20 FF AE   JSR $AEFF    prüft auf Code
ABCB   20 21 AB   JSR $AB21    String ausgeben

Einsprung von $ABB2

ABCE   20 A6 B3   JSR $B3A6    prüft auf Direktmodus
ABD1   A9 2C      LDA #$2C     ','  Komma
ABD3   8D FF 01   STA $01FF    an Pufferstart
ABD6   20 F9 AB   JSR $ABF9    Fragezeichen ausgeben
ABD9   A5 13      LDA $13      Nummer des Eingabegeräts
ABDB   F0 0D      BEQ $ABEA    Tastatur? ja: $ABEA
ABDD   20 B7 FF   JSR $FFB7    Status holen
ABE0   29 02      AND #$02     Bit 1 isolieren (Tineout R.)
ABE2   F0 06      BEQ $ABEA    Time-out?
ABE4   20 B5 AB   JSR $ABB5    ja: CLRCH,Tastatur aktivieren
ABE7   4C F8 A8   JMP $A8F8    nächstes Statement ausführen

ABEA   AD 00 02   LDA $0200    erstes Zeichen holen
ABED   D0 1E      BNE $AC0D    Ende?
ABEF   A5 13      LDA $13      ja: Eingabegerät
ABF1   D0 E3      BNE $ABD6    nicht Tastatur: $ABD6
ABF3   20 06 A9   JSR $A906    Offset (Statement) suchen
ABF6   4C FB A8   JMP $A8FB    Programmzeiger auf Statement

Einsprung von $ABD6, $AC4A

ABF9   A5 13      LDA $13      Eingabegerät holen
ABFB   D0 06      BNE $AC03    nicht Tastatur: $AC03
ABFD   20 45 AB   JSR $AB45    '?' ausgeben
AC00   20 3B AB   JSR $AB3B    ' ' Leerzeichen ausgeben
AC03   4C 60 A5   JMP $A560    Eingabezeile holen

*****************************  BASIC-Befehl  READ
AC06   A6 41      LDX $41      DATA-Zeiger nach
AC08   A4 42      LDY $42      $41/42 holen
AC0A   A9 98      LDA #$98     READ-Flag
AC0C   2C         .BYTE $2C
AC0D   A9 00      LDA #$00     Flag-Wert laden

Einsprung von $AB9D

AC0F   85 11      STA $11      und INPUT-Zeiger setzen
AC11   86 43      STX $43      INPUT-Zeiger auf
AC13   84 44      STY $44      Eingabequelle setzen

Einsprung von $ACB5

AC15   20 8B B0   JSR $B08B    sucht Variable
AC18   85 49      STA $49      Variablenadresse
AC1A   84 4A      STY $4A      speichern
AC1C   A5 7A      LDA $7A      Low- und High-Byte des
```

```
AC1E   A4 7B      LDY $7B     Programmzeigers
AC20   85 4B      STA $4B     in $4B/$4C
AC22   84 4C      STY $4C     zwischenspeichern
AC24   A6 43      LDX $43     INPUT-Zeiger
AC26   A4 44      LDY $44     (Low und High)
AC28   86 7A      STX $7A     als Programmzeiger
AC2A   84 7B      STY $7B     abspeichern
AC2C   20 79 00   JSR $0079   CHRGOT letztes Zeichen holen
AC2F   D0 20      BNE $AC51   Endzeichen? Nein: $AC51
AC31   24 11      BIT $11     Eingabe-Flag
AC33   50 0C      BVC $AC41   kein GET: $AC41
AC35   20 24 E1   JSR $E124   GETIN
AC38   8D 00 02   STA $0200   Zeichen in Puffer schreiben
AC3B   A2 FF      LDX #$FF    Zeiger auf
AC3D   A0 01      LDY #$01    Puffer setzen
AC3F   D0 0C      BNE $AC4D   unbedingter Sprung
AC41   30 75      BMI $ACB8   READ: $ACB8
AC43   A5 13      LDA $13     Eingabegerät holen
AC45   D0 03      BNE $AC4A   nicht Tastatur: $AC4A
AC47   20 45 AB   JSR $AB45   Fragezeichen ausgeben
AC4A   20 F9 AB   JSR $ABF9   zweites Fragezeichen ausgeben
AC4D   86 7A      STX $7A     Programmzeiger setzen
AC4F   84 7B      STY $7B     (Low und High)
```

Einsprung von $ACDC

```
AC51   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
AC54   24 0D      BIT $0D     Typ-Flag
AC56   10 31      BPL $AC89   Kein String: $AC89
AC58   24 11      BIT $11     Eingabe-Flag
AC5A   50 09      BVC $AC65   kein GET: $AC65
AC5C   E8         INX         Programmzeiger erhöhen
AC5D   86 7A      STX $7A     und neu setzen ($0200)
AC5F   A9 00      LDA #$00    Wert laden und
AC61   85 07      STA $07     Trennzeichen setzen
AC63   F0 0C      BEQ $AC71   unbedingter Sprung
AC65   85 07      STA $07     nächstes Zeichen
AC67   C9 22      CMP #$22    '"' Hochkomma?
AC69   F0 07      BEQ $AC72   ja: $AC72
AC6B   A9 3A      LDA #$3A    ':' Doppelpunktcode laden
AC6D   85 07      STA $07     und abspeichern
AC6F   A9 2C      LDA #$2C    ',' Kommacode (Endzeichen
AC71   18         CLC         für String-Übertragung)
AC72   85 08      STA $08     abspeichern
AC74   A5 7A      LDA $7A     Programmzeiger laden
AC76   A4 7B      LDY $7B     (Low und High)
AC78   69 00      ADC #$00    und Übertrag addieren
AC7A   90 01      BCC $AC7D   C = 0: $AC7D
AC7C   C8         INY         bei '"' um 1 erhöhen
AC7D   20 8D B4   JSR $B48D   String übernehmen
AC80   20 E2 B7   JSR $B7E2   Programmzeiger hinter String
AC83   20 DA A9   JSR $A9DA   String an Variable zuweisen
AC86   4C 91 AC   JMP $AC91   weiter machen
```

```
ACB9   20 F3 BC   JSR $BCF3     Ziffern-String in FAC holen
ACBC   A5 0E      LDA $0E       INTEGER/REAL-Flag
ACBE   20 C2 A9   JSR $A9C2     FAC an numerische Variable
```

Einsprung von $AC86

```
AC91   20 79 00   JSR $0079     CHRGOT: letztes Zeichen holen
AC94   F0 07      BEQ $AC9D     Ende?
AC96   C9 2C      CMP #$2C      ',' Code?
AC98   F0 03      BEQ $AC9D     ja: $AC9D
AC9A   4C 4D AB   JMP $AB4D     zur Fehlerbehandlung
AC9D   A5 7A      LDA $7A       Programmzeiger
AC9F   A4 7B      LDY $7B       holen und
ACA1   85 43      STA $43       in DATA-Zeiger
ACA3   84 44      STY $44       abspeichern
ACA5   A5 4B      LDA $4B       ursprüngliche
ACA7   A4 4C      LDY $4C       Programmzeiger
ACA9   85 7A      STA $7A       wieder zurückholen
ACAB   84 7B      STY $7B       und speichern
ACAD   20 79 00   JSR $0079     CHRGOT: letztes Zeichen holen
ACB0   F0 2D      BEQ $ACDF     Trennzeichen: $ACDF
ACB2   20 FD AE   JSR $AEFD     CKCOM: prüft auf Komma
ACB5   4C 15 AC   JMP $AC15     weiter
ACB8   20 06 A9   JSR $A906     nächstes Statement suchen
ACBB   C8         INY           Offset erhöhen
ACBC   AA         TAX           Zeilenende?
ACBD   D0 12      BNE $ACD1     Nein: $ACD1
ACBF   A2 0D      LDX #$0D      'OUT OF DATA' Code
ACC1   C8         INY           Zeiger erhöhen
ACC2   B1 7A      LDA ($7A),Y   Programmende?
ACC4   F0 6C      BEQ $AD32     Ja: 'OUT OF DATA', X = 0
ACC6   C8         INY           Zeiger erhöhen
ACC7   B1 7A      LDA ($7A),Y   Zeilennummer (Low) holen
ACC9   85 3F      STA $3F       und abspeichern
ACCB   C8         INY           Zeiger erhöhen
ACCC   B1 7A      LDA ($7A),Y   Zeilenummer (High)
ACCE   C8         INY           Zeiger erhöhen
ACCF   85 40      STA $40       Zeilennummer speichern
ACD1   20 FB A8   JSR $A8FB     Programmz. auf Statement
ACD4   20 79 00   JSR $0079     CHRGOT letztes Zeichen holen
ACD7   AA         TAX           und ins X-Reg.
ACD8   E0 83      CPX #$83      'DATA' Code?
ACDA   D0 DC      BNE $ACB8     Nein: weitersuchen
ACDC   4C 51 AC   JMP $AC51     Daten lesen
ACDF   A5 43      LDA $43       Low- und High-Byte des
ACE1   A4 44      LDY $44       Input-Zeigers
ACE3   A6 11      LDX $11       Eingabe-Flag
ACE5   10 03      BPL $ACEA     kein DATA: $ACEA
ACE7   4C 27 A8   JMP $A827     DATA-Zeiger setzen
ACEA   A0 00      LDY #$00      Zeiger setzen
ACEC   B1 43      LDA ($43),Y   nächstes Zeichen holen
ACEE   F0 0B      BEQ $ACFB     Endzeichen: $ACFB
ACF0   A5 13      LDA $13       Eingabe über Tastatur?
```

```
ACF2    D0 07       BNE $ACFB    Nein: $ACFB
ACF4    A9 FC       LDA #$FC     Zeiger auf
ACF6    A0 AC       LDY #$AC     '?extra ignored' setzen
ACF8    4C 1E AB    JMP $AB1E    String ausgeben
ACFB    60          RTS          Rücksprung

ACFC    3F 45 58 54 52 41 20 49   '?extra ignored'
AD04    47 4E 4F 52 45 44 00 00
AD0C    3F 52 45 44 4F 20 46 52   '?redo from start'
AD14    4F 20 53 54 41 52 54 0D
AD1C    00

****************************     BASIC-Befehl  NEXT
AD1D    D0 04       BNE $AD24    folgt Variablenname? ja:$AD24
AD20    A0 00       LDY #$00     Variablenzeiger = 0
AD22    F0 03       BEQ $AD27    unbedingter Sprung

Einsprung von $AD87

AD24    20 8B B0    JSR $B08B    sucht Variable
AD27    85 49       STA $49      Adresse der
AD29    84 4A       STY $4A      Variablen speichern
AD2B    20 8A A3    JSR $A38A    sucht FOR-NEXT-Schleife
AD2E    F0 05       BEQ $AD35    gefunden: $AD35
AD30    A2 0A       LDX #$0A     Nummer für 'next without for'
AD32    4C 37 A4    JMP $A437    Fehlermeldung ausgeben
AD35    9A          TXS          X-Reg. retten
AD36    8A          TXA          X-Register nach Akku
AD37    18          CLC          Carry löschen (Addition)
AD38    69 04       ADC #$04     Zeiger auf Exponenten des
AD3A    48          PHA          STEP-Wert + 4 und retten
AD3B    69 06       ADC #$06     Zeiger auf Exponent des TO-
AD3D    85 24       STA $24      Wert und retten
AD3F    68          PLA          Akku wieder vom Stapel holen
AD40    A0 01       LDY #$01     Zeiger für Konstante setzen
AD42    20 A2 BB    JSR $BBA2    Variable vom Stapel nach FAC
AD45    BA          TSX          Stapelzeiger als Zeiger h.
AD46    BD 09 01    LDA $0109,X  Vorzeichen-Byte holen und
AD49    85 66       STA $66      für FAC speichern
AD4B    A5 49       LDA $49      Variablenadresse für
AD4D    A4 4A       LDY $4A      FOR-NEXT holen
AD4F    20 67 B8    JSR $B867    addiert STEP-Wert zu FAC
AD52    20 D0 BB    JSR $BBD0    FAC nach Variable bringen
AD55    A0 01       LDY #$01     Zeiger auf Konstante setzen
AD57    20 5D BC    JSR $BC5D    FAC mit Schleifenendwert vergleichen
AD5A    BA          TSX          Stapelzeiger als Zeiger h.
AD5B    38          SEC          Carry setzen (Subtraktion)
AD5C    FD 09 01    SBC $0109,X  Stapelwert größer?
AD5F    F0 17       BEQ $AD78    Ja: Schleife verlassen
AD61    BD 0F 01    LDA $010F,X  Zeilennummer des Schleifen-
AD64    85 39       STA $39      anfangs holen (Low- und
AD66    BD 10 01    LDA $0110,X  High-Byte) und als aktuelle
AD69    85 3A       STA $3A      BASIC-Zeilennummer speichern
```

```
AD6B   BD 12 01   LDA $0112,X Schleifenanfang holen (Low-
AD6E   85 7A      STA $7A     und High-Byte) und
AD70   BD 11 01   LDA $0111,X als neuen Programmzeiger
AD73   85 7B      STA $7B     abspeichern
AD75   4C AE A7   JMP $A7AE   zur Interpreter-Schleife
AD78   8A         TXA         Zeiger in Akku holen
AD79   69 11      ADC #$11    (Werte der Schleife aus
AD7B   AA         TAX         Stapel entfernen)
AD7C   9A         TXS         neuen Stapelzeiger setzen
AD7D   20 79 00   JSR $0079   CHRGOT letztes Zeichen holen
AD80   C9 2C      CMP #$2C    ',' Komma?
AD82   D0 F1      BNE $AD75   Nein: dann fertig
AD84   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
AD87   20 24 AD   JSR $AD24   nächste  NEXT-Variable

****************************  FRMNUM  Ausdruck holen und
                              auf numerisch prüfen

Einsprung von $A775, $A79C, $B438, $B79E, B7EB, $E12A

AD8A   20 9E AD   JSR $AD9E   FRMEVL  Term holen

****************************  prüft auf numerisch

Einsprung von $A772, $ADF6, $AE61, $AFE3, B1B8, $B3C3
$B3F1, $B400, $B465

AD8D   18         CLC         Flag für Test auf numerisch
AD8E   24         .BYTE $24

****************************  prüft auf String

Einsprung von $AFBA, $B646, $B6A3

AD8F   38         SEC         Flag für Test auf String

Einsprung von $A9BC, $B016

AD90   24 0D      BIT $0D     Typ-Flag testen
AD92   30 03      BMI $AD97   gesetzt: $AD97
AD94   B0 03      BCS $AD99   C=1: 'TYPE MISMATCH'
AD96   60         RTS         Rücksprung
AD97   B0 FD      BCS $AD96   C=1: RTS
AD99   A2 16      LDX #$16    Nummer für 'TYPE MISMATCH'
AD9B   4C 37 A4   JMP $A437   Fehlermeldung ausgeben

****************************  FRMEVL  auswerten eines
                              beliebigen Ausdrucks

Einsprung von $A928, $A9B7, $AAB5, $AD8A, $AEF4, $AFB4
$B1B5, $E257
```

```
AD9E    A6 7A       LDX $7A       Programmzeiger (Low) = 0?
ADA0    D0 02       BNE $ADA4     ja: High-B. nicht vermindern
ADA2    C6 7B       DEC $7B       High-Byte vermindern
ADA4    C6 7A       DEC $7A       Low-Byte vermindern
ADA6    A2 00       LDX #$00      Prioritätswert laden
ADA8    24          .BYTE $24
```

Einsprung von $AE2D

```
ADA9    48          PHA           Operatormaske retten
ADAA    8A          TXA           Prioritätswert in Akku
ADAB    48          PHA           schieben und retten
ADAC    A9 01       LDA #$01      2 Bytes
ADAE    20 FB A3    JSR $A3FB     prüft auf Platz im Stapel
ADB1    20 83 AE    JSR $AE83     Nächstes Element holen
ADB4    A9 00       LDA #$00      Wert laden und
ADB6    85 4D       STA $4D       Maske für Vergleichsoperator
```

Einsprung von $B677

```
ADB8    20 79 00    JSR $0079     CHRGOT letztes Zeichen holen
```

Einsprung von $ADD4

```
ADBB    38          SEC           Carry setzen (Subtraktion)
ADBC    E9 B1       SBC #$B1      $B1 von Operatorcode subtr.
ADBE    90 17       BCC $ADD7     C=0: $ADD7
ADC0    C9 03       CMP #$03      mit $3 vergleichen
ADC2    B0 13       BCS $ADD7     =3: $ADD7
ADC4    C9 01       CMP #$01
ADC6    2A          ROL           Maske für kleiner
ADC7    49 01       EOR #$01      gleich und größer
ADC9    45 4D       EOR $4D       für Bits 0,1 und 2
ADCB    C5 4D       CMP $4D       in $4D erstellen
ADCD    90 61       BCC $AE30     (Wenn Codes von 177
ADCF    85 4D       STA $4D       bis 179 folgen)
ADD1    20 73 00    JSR $0073     CHRGET nächstes Zeichen holen
ADD4    4C BB AD    JMP $ADBB     nächstes Zeichen auswerten
ADD7    A6 4D       LDX $4D       Operatormaske holen
ADD9    D0 2C       BNE $AE07     gleich 0? nein: $AE07
ADDB    B0 7B       BCS $AE58     Code größer oder gleich 180?
ADDD    69 07       ADC #$07      Code kleiner 170?
ADDF    90 77       BCC $AE58     ja: $AE58
ADE1    65 0D       ADC $0D       String-Addition?
ADE3    D0 03       BNE $ADE8     Nein: Verkettung umgehen
ADE5    4C 3D B6    JMP $B63D     String-Verkettung
ADE8    69 FF       ADC #$FF      Code-$AA (wiederherstellen)
ADEA    85 22       STA $22       und speichern
ADEC    0A          ASL           verdoppeln
ADED    65 22       ADC $22       + Wert (also mal 3)
ADEF    A8          TAY           als Zeiger ins Y-Register
ADF0    68          PLA           bisheriger Prioritätswert
ADF1    D9 80 A0    CMP $A080,Y   mit Prioritätsw. vergleichen
```

```
ADF4   B0 67      BCS $AE5D   größer: $AE5D
ADF6   20 8D AD   JSR $AD8D   prüft auf numerisch
ADF9   48         PHA         Prioritätswert retten
```

Einsprung von $AF11

```
ADFA   20 20 AE   JSR $AE20   Operatoradr. und Operanden r.
ADFD   68         PLA
ADFE   A4 4B      LDY $4B     Operator?
AE00   10 17      BPL $AE19   Ja: $AE19
AE02   AA         TAX         weitere Operation?
AE03   F0 56      BEQ $AE5B   Nein: RTS
AE05   D0 5F      BNE $AE66   ARG vom Stapel holen
AE07   46 0D      LSR $0D     String-Flag löschen
AE09   8A         TXA         Operatormaske nach
AE0A   2A         ROL         links schieben
AE0B   A6 7A      LDX $7A     Programmzeiger holen (Low)
AE0D   D0 02      BNE $AE11   =0: High-Byte vermindern
AE0F   C6 7B      DEC $7B     High-Byte vermindern
AE11   C6 7A      DEC $7A     Low-Byte vermindern
AE13   A0 1B      LDY #$1B    Offset des Hierarchie-Flags
AE15   85 4D      STA $4D     Flag setzen
AE17   D0 D7      BNE $ADF0   unbedingter Sprung
AE19   D9 80 A0   CMP $A080,Y mit Hierarchie-Flag vergl.
AE1C   B0 48      BCS $AE66   größer: $AE66
AE1E   90 D9      BCC $ADF9   sonst weiter
```

Einsprung von $ADFA

```
AE20   B9 82 A0   LDA $A082,Y Operationsadresse (High)
AE23   48         PHA         auf Stapel retten
AE24   B9 81 A0   LDA $A081,Y Operationsadresse (Low)
AE27   48         PHA         auf Stapel retten
AE28   20 33 AE   JSR $AE33   Operanden auf Stapel retten
AE2B   A5 4D      LDA $4D     Operatormaske laden
AE2D   4C A9 AD   JMP $ADA9   zum Schleifenanfang
AE30   4C 08 AF   JMP $AF08   gibt 'SYNTAX ERROR'
```

Einsprung von $AE28

```
AE33   A5 66      LDA $66     Vorzeichen von FAC
AE35   BE 80 A0   LDX $A080,Y Hierarchie-Flag
```

Einsprung von $A7A2

```
AE38   A8         TAY         Vorzeichen ins Y-Reg.
AE39   68         PLA         Rücksprungadresse holen
AE3A   85 22      STA $22     und merken
AE3C   E6 22      INC $22     Rücksprungadresse erhöhen
AE3E   68         PLA         nächstes Adreß-Byte holen
AE3F   85 23      STA $23     und speichern
```

```
AE41   98         TYA          Vorzeichen wieder in Akku
AE42   48         PHA          und auf Stapel legen

Einsprung von $A788

AE43   20 1B BC   JSR $BC1B    FAC runden
AE46   A5 65      LDA $65      FAC auf Stapel legen
AE48   48         PHA          1. Byte retten
AE49   A5 64      LDA $64      2. Byte holen
AE4B   48         PHA          und retten
AE4C   A5 63      LDA $63      3. Byte holen
AE4E   48         PHA          und retten
AE4F   A5 62      LDA $62      4. Byte holen
AE51   48         PHA          und retten
AE52   A5 61      LDA $61      5. Byte holen
AE54   48         PHA          und retten
AE55   6C 22 00   JMP ($0022)  Sprung auf Operation
AE58   A0 FF      LDY #$FF     Flag-Wert für Operator
AE5A   68         PLA          Prioritäts-Flag retten
AE5B   F0 23      BEQ $AE80    =0? Ja: $AE80
AE5D   C9 64      CMP #$64     =$64?
AE5F   F0 03      BEQ $AE64    Ja: $AE64
AE61   20 8D AD   JSR $AD8D    prüft auf numerisch
AE64   84 4B      STY $4B      Flag für Operator
AE66   68         PLA          Akku vom Stapel holen
AE67   4A         LSR          halbieren
AE68   85 12      STA $12      und abspeichern
AE6A   68         PLA          ARG von Stapel holen
AE6B   85 69      STA $69      1. Byte speichern
AE6D   68         PLA          2. Byte holen
AE6E   85 6A      STA $6A      und speichern
AE70   68         PLA          3. Byte holen
AE71   85 6B      STA $6B      und speichern
AE73   68         PLA          4. Byte holen
AE74   85 6C      STA $6C      und speichern
AE76   68         PLA          5. Byte holen
AE77   85 6D      STA $6D      und speichern
AE79   68         PLA          6. Byte (Vorzeichen holen
AE7A   85 6E      STA $6E      und speichern
AE7C   45 66      EOR $66      Vorzeichen von ARG und FAC
AE7E   85 6F      STA $6F      verknüpfen und speichern
AE80   A5 61      LDA $61      Exponent-Byte von FAC laden
AE82   60         RTS          Rücksprung

*****************************  Nächstes Element eines
                               Ausdrucks holen

Einsprung von $ADB1, $B643

AE83   6C 0A 03   JMP ($030A) JMP $AE86

Einsprung von $AE83
```

```
AE86   A9 00      LDA #$00    Wert laden und damit
AE88   85 0D      STA $0D     Typ-Flag auf numerisch setzen
AE8A   20 73 00   JSR $0073   CHRGET nächstes Zeichen holen
AE8D   B0 03      BCS $AE92   Ziffer? Nein: $AE92
AE8F   4C F3 BC   JMP $BCF3   Variable nach FAC holen
AE92   20 13 B1   JSR $B113   Buchstabe?
AE95   90 03      BCC $AE9A   Nein: JMP umgehen
AE97   4C 28 AF   JMP $AF28   Variable holen
AE9A   C9 FF      CMP #$FF    BASIC-Code für Pi?
AE9C   D0 0F      BNE $AEAD   Nein: $AEAD
AE9E   A9 A8      LDA #$A8    Zeiger auf Konstante Pi
AEA0   A0 AE      LDY #$AE    (Low und High-Byte)
AEA2   20 A2 BB   JSR $BBA2   Konstante in FAC holen
AEA5   4C 73 00   JMP $0073   CHRGET nächstes Zeichen holen

*****************************
AEA8   82 49 0F DA A1         Konstante Pi  3.14159265

*****************************
AEAD   C9 2E      CMP #$2E    '.' Dezimalpunkt?
AEAF   F0 DE      BEQ $AE8F   Ja: $AE8F
AEB1   C9 AB      CMP #$AB    '-'?
AEB3   F0 58      BEQ $AF0D   zum Vorzeichenwechsel
AEB5   C9 AA      CMP #$AA    '+'?
AEB7   F0 D1      BEQ $AE8A   Ja: $Ae8A
AEB9   C9 22      CMP #$22    '"'?
AEBB   D0 0F      BNE $AECC   Nein: $AECC
AEBD   A5 7A      LDA $7A     Low- und High-Byte des
AEBF   A4 7B      LDY $7B     Programmzeigers holen
AEC1   69 00      ADC #$00    und Übertrag addieren
AEC3   90 01      BCC $AEC6   C=0: $AEC6
AEC5   C8         INY         High-Byte erhöhen
AEC6   20 87 B4   JSR $B487   String übertragen
AEC9   4C E2 B7   JMP $B7E2   Programmz. auf Stringende +1
AECC   C9 A8      CMP #$A8    'NOT'-Code?
AECE   D0 13      BNE $AEE3   Nein: $AEE3
AED0   A0 18      LDY #$18    Offset des H.Flags in Tabelle
AED2   D0 3B      BNE $AF0F   unbedingter Sprung

*****************************  BASIC-Befehl  NOT
AED4   20 BF B1   JSR $B1BF   FAC nach INTEGER wandeln
AED7   A5 65      LDA $65     High-Byte holen
AED9   49 FF      EOR #$FF    alle Bits umdrehen
AEDB   A8         TAY         und ins Y-Reg.
AEDC   A5 64      LDA $64     Low-Byte holen
AEDE   49 FF      EOR #$FF    alle Bits invertieren
AEE0   4C 91 B3   JMP $B391   nach Fließkomma wandeln

*****************************
AEE3   C9 A5      CMP #$A5    'FN'-Code?
AEE5   D0 03      BNE $AEEA   Nein: $AEEA
AEE7   4C F4 B3   JMP $B3F4   FN ausführen
```

```
******************************
AEEA   C9 B4      CMP #$B4    'SGN'-Code
AEEC   90 03      BCC $AEF1   kleiner (keine String-Funkt.)?
AEEE   4C A7 AF   JMP $AFA7   holt String ,ersten Parameter

******************************  holt Term in Klammern

Einsprung von $B3FD

AEF1   20 FA AE   JSR $AEFA   prüft auf Klammer auf
AEF4   20 9E AD   JSR $AD9E   FRMEVL  holt Term

******************************  prüft auf Zeichen im B.-Text

Einsprung von $B20B, $B3C6, $B761

AEF7   A9 29      LDA #$29    ')'   Klammer zu
AEF9   2C         .BYTE $2C

Einsprung von $AEF1, $AFB1, $B3B9

AEFA   A9 28      LDA #$28    '('   Klammer auf
AEFC   2C         .BYTE $2C

Einsprung von $ACB2, $AFB7, $B07E, $B742, B7F1, $E20E

AEFD   A9 2C      LDA #$2C    ','   Komma

Einsprung von $A76F, $A817, $A934, $A9AE, $AA8D, AB8A
$ABAA, $ABC8, $B3CB, $B3E3

AEFF   A0 00      LDY #$00    Zeiger setzen
AF01   D1 7A      CMP ($7A),Y mit laufendem Zeichen vergl.
AF03   D0 03      BNE $AF08   keine Übereinstimmung?
AF05   4C 73 00   JMP $0073   CHRGET nächstes Zeichen holen
Einsprung von $A80B, $A8E8, $AB5F, $AE30, $B09C, $B138
$B446, $E216

AF08   A2 0B      LDX #$0B    Nummer für 'SYNTAX ERROR'
AF0A   4C 37 A4   JMP $A437   Fehlermeldung ausgeben

AF0D   A0 15      LDY #$15    Offset Hierachie-Code für VZW
AF0F   68         PLA         nächsten 2 Bytes vom
AF10   68         PLA         Stapel entfernen
AF11   4C FA AD   JMP $ADFA   zur Auswertung

******************************  prüft auf Variable
                                innerhalb des BASICs

Einsprung von $AF3B, $AF6E

AF14   38         SEC         Carry setzen (Subtr.)
AF15   A5 64      LDA $64     Descriptor holen
```

```
AF17   E9 00      SBC #$00      liegt Descriptor ($64/$65)
AF19   A5 65      LDA $65       zwischen $A000 und $E32A?
AF1B   E9 A0      SBC #$A0
AF1D   90 08      BCC $AF27     Ja: dann C=1, sonst RTS
AF1F   A9 A2      LDA #$A2      1. Wert laden
AF21   E5 64      SBC $64       1. Descriptor-Byte abziehen
AF23   A9 E3      LDA #$E3      2. Wert laden
AF25   E5 65      SBC $65       und Descriptorwert abziehen
AF27   60         RTS           Rücksprung

******************************  Variable holen

Einsprung von $AE97

AF28   20 8B B0   JSR $B08B     Variable suchen
AF2B   85 64      STA $64       Zeiger auf Variable
AF2D   84 65      STY $65       bzw. String-Descriptor
AF2F   A6 45      LDX $45       als
AF31   A4 46      LDY $46       Variablenname speichern
AF33   A5 0D      LDA $0D       Typ-Flag holen
AF35   F0 26      BEQ $AF5D     numerisch?
AF37   A9 00      LDA #$00      Wert laden und
AF39   85 70      STA $70       in Rundungs-Byte für FAC
AF3B   20 14 AF   JSR $AF14     Descriptor im Interpreter?
AF3E   90 1C      BCC $AF5C     Nein
AF40   E0 54      CPX #$54      'T'? (von TI$)
AF42   D0 18      BNE $AF5C     Nein: $AF5C
AF44   C0 C9      CPY #$C9      'I$'? (von TI$)
AF46   D0 14      BNE $AF5C     Nein: $AF5C
AF48   20 84 AF   JSR $AF84     Zeit nach FAC holen
AF4B   84 5E      STY $5E       Flag für Exponentialdarst. =0
AF4D   88         DEY           vermindern (=$FF)
AF4E   84 71      STY $71       Zeiger für String-Startadresse
AF50   A0 06      LDY #$06      Länge 6 für TI$
AF52   84 5D      STY $5D       speichern
AF54   A0 24      LDY #$24      Zeiger auf Stellenwert
AF56   20 68 BE   JSR $BE68     erzeugt String TI$
AF59   4C 6F B4   JMP $B46F     bringt String in Str.bereich
AF5C   60         RTS           Rücksprung

AF5D   24 0E      BIT $0E       INTEGER/ REAL Flag
AF5F   10 0D      BPL $AF6E     REAL? Ja: $AF6E

******************************  Integer-Variable holen
AF61   A0 00      LDY #$00      Zeiger setzen
AF63   B1 64      LDA ($64),Y   Integer-Zahl holen (1. Byte)
AF65   AA         TAX           ins X-Reg.
AF66   C8         INY           Zeiger erhöhen
AF67   B1 64      LDA ($64),Y   2. Byte holen
AF69   A8         TAY           ins Y-Register
AF6A   8A         TXA           1. Byte in Akku holen
AF6B   4C 91 B3   JMP $B391     und nach Fließkomma wandeln
```

```
*****************************  REAL-Variable holen
AF6E  20 14 AF   JSR $AF14     Descriptor im Interpreter?
AF71  90 2D      BCC $AFA0     Nein
AF73  E0 54      CPX #$54      'T'? (von TI)
AF75  D0 1B      BNE $AF92     Nein: $AF92
AF77  C0 49      CPY #$49      'I'? (von TI)
AF79  D0 25      BNE $AFA0     Nein: $AFA0
AF7B  20 84 AF   JSR $AF84     TIME in FAC holen
AF7E  98         TYA           Akku =0 setzen
AF7F  A2 A0      LDX #$A0      Exponent-Byte für FAC
AF81  4C 4F BC   JMP $BC4F     FAC linksbündig machen

*****************************  Zeit holen

Einsprung von $AF48, $AF7B

AF84  20 DE FF   JSR $FFDE     TIME holen
AF87  86 64      STX $64       1. Byte nach FAC
AF89  84 63      STY $63       2. Byte nach FAC
AF8B  85 65      STA $65       3. Byte nach FAC
AF8D  A0 00      LDY #$00      Wert laden (0) und
AF8F  84 62      STY $62       als 4. Byte nach FAC
AF91  60         RTS           Rücksprung

AF92  E0 53      CPX #$53      'S'?
AF94  D0 0A      BNE $AFA0     Nein: $AFA0
AF96  C0 54      CPY #$54      'T'?
AF98  D0 06      BNE $AFA0     Nein: $AFA0
AF9A  20 B7 FF   JSR $FFB7     Status holen
AF9D  4C 3C BC   JMP $BC3C     Byte in Fließkommaformat

*****************************  REAL-Variable holen
AFA0  A5 64      LDA $64       Low- und High-Byte der
AFA2  A4 65      LDY $65       Variablenadresse
AFA4  4C A2 BB   JMP $BBA2     Variable in FAC holen

*****************************  Funktionsberechnung

Einsprung von $AEEE

AFA7  0A         ASL           Funktionscode mal 2
AFA8  48         PHA           auf den Stapel retten
AFA9  AA         TAX           und ins X-Register
AFAA  20 73 00   JSR $0073     CHRGET nächstes Zeichen
AFAD  E0 8F      CPX #$8F      Numerische Funktion?
AFAF  90 20      BCC $AFD1     Ja: $AFD1

*****************************  Stringfunktion, String und
                               ersten Parameter
AFB1  20 FA AE   JSR $AEFA     prüft auf Klammer auf
AFB4  20 9E AD   JSR $AD9E     FRMEVL holen beliebigen Term
AFB7  20 FD AE   JSR $AEFD     prüft auf Komma
AFBA  20 8F AD   JSR $AD8F     prüft auf String
```

```
AFBD  68        PLA          Funktionstoken left$, r$, m$
AFBE  AA        TAX          Akku nach X holen
AFBF  A5 65     LDA $65      Adresse des
AFC1  48        PHA          Stringdescriptors
AFC2  A5 64     LDA $64      holen und auf den Stapel
AFC4  48        PHA          retten (Low und High)
AFC5  8A        TXA          Akku wiederholen
AFC6  48        PHA          Token auf den Stapel retten
AFC7  20 9E B7  JSR $B79E    holt Byte-Wert (2. Parameter)
AFCA  68        PLA          Token zurückholen
AFCB  A8        TAY          und ins Y-Reg.
AFCC  8A        TXA          2. Byte-Wert in den Akku laden
AFCD  48        PHA          und auf den Stapel retten
AFCE  4C D6 AF  JMP $AFD6    Routine ausführen

****************************** numerische Funktion auswerten
AFD1  20 F1 AE  JSR $AEF1    holt Term in Klammern
AFD4  68        PLA          BASIC-Code für Funktion holen
AFD5  A8        TAY          und als Zeiger ins Y-Reg.

Einsprung von $AFCE

AFD6  B9 EA 9F  LDA $9FEA,Y  Vektor für Funktionsbe-
AFD9  85 55     STA $55      rechnung holen und speichern
AFDB  B9 EB 9F  LDA $9FEB,Y  2.Byte holen
AFDE  85 56     STA $56      und speichern
AFE0  20 54 00  JSR $0054    Funktion ausführen
AFE3  4C 8D AD  JMP $AD8D    prüft auf numerisch

****************************** BASIC-Befehl  OR
AFE6  A0 FF     LDY #$FF     Flag für OR
AFE8  2C        .BYTE $2C

****************************** BASIC-Befehl  AND
AFE9  A0 00     LDY #$00     Flag für AND
AFEB  84 0B     STY $0B      Flag setzen
AFED  20 BF B1  JSR $B1BF    FAC nach INTEGER wandeln
AFF0  A5 64     LDA $64      ersten Wert holen
AFF2  45 0B     EOR $0B      mit Flag verknüpfen
AFF4  85 07     STA $07      und speichern
AFF6  A5 65     LDA $65      zweiten Wert holen
AFF8  45 0B     EOR $0B      mit Flag verknüpfen
AFFA  85 08     STA $08      und speichern
AFFC  20 FC BB  JSR $BBFC    ARG nach FAC
AFFF  20 BF B1  JSR $B1BF    FAC nach Integer
```

Anhang C.10: Der Schaltplan

Zu Anfang dieses Anhangs einige Vorbemerkungen: Leider können die folgenden Seiten keine Einführung in die Digital- oder Computertechnik bieten.

Wir müssen einige elementare Kenntnisse dieser Technik voraussetzen. So sollten Sie den Unterschied zwischen einem AND- und einem OR-Gate kennen oder sich beispielsweise in der Benutzung der Hexadezimalzahlen auskennen. Wenn Sie diese Grundkenntnisse bereits haben, bisher aber mit der Hardware von Microcomputern nichts zu tun hatten, so sollten Sie sich von der etwas verwirrenden Anzahl der Leitungen, Gatter und anderen ICs im Schaltplan nicht beeindrucken lassen.

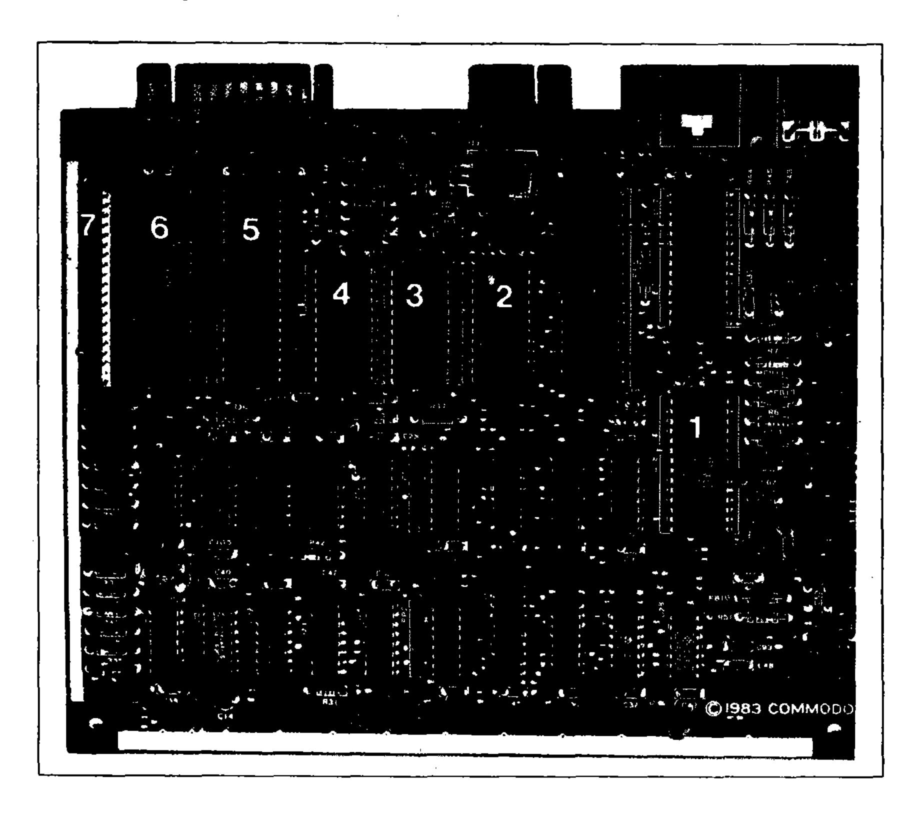
Nach der Lektüre dieses Anhangs werden Sie die Hardware Ihres Computers recht gut verstehen. Den Spezialisten und "Freaks" unter Ihnen wird die Beschreibung sicher zu ausführlich erscheinen. Sie sollten dieses Kapitel trotzdem in Ruhe durchlesen. Um die Funktionen der einzelnen Stufen nur an Hand des Schaltplans im Detail zu verstehen, ist wesentlich mehr Zeit erforderlich, als zum Lesen dieses Kapitels benötigt wird.

Jeder technisch interessierte Computerbesitzer hat sicher den Wunsch, sein Gerät einmal zu öffnen und hineinzuschauen. Vielleicht haben auch Sie schon einmal das Innenleben betrachtet. Sollten Sie aber aus Vorsicht, den C64 nicht zu beschädigen, diesem Wunsch nicht nachgegeben haben, dann seien Sie beruhigt. Lösen Sie zuerst alle Leitungen zum C64, also Netzteil, Fernseher und alle anderen angeschlossenen Geräte. Dann können Sie unbesorgt zu einem passenden Kreuzschlitzschraubenzieher greifen, die auf der Unterseite befindlichen drei Schrauben lösen und vorsichtig die beiden Gehäusehälften trennen, um einen Blick in den Computer zu werfen. Lösen Sie aber unbedingt zuvor die Steckverbindung zum Netzteil, um versehentliche Kurzschlüsse zu vermeiden.

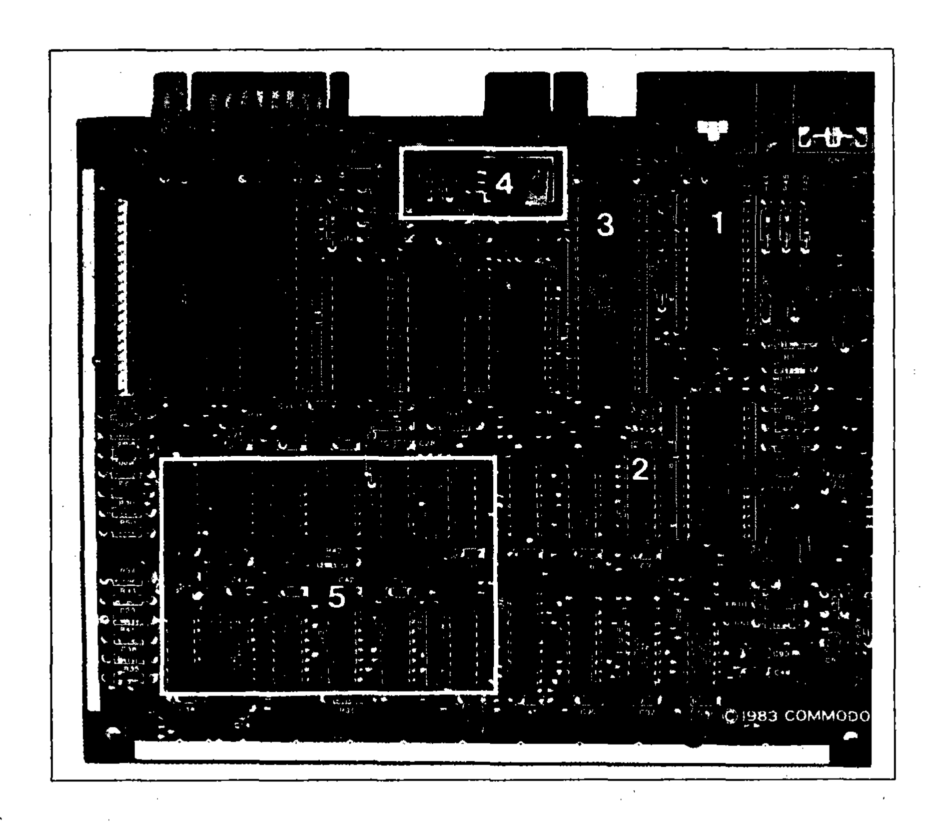
Das Foto zeigt den Blick in einen C64. Wenn Sie feststellen, daß Ihr C64 ein etwas anderes Innenleben aufweist, dann liegt das daran, daß das Layout der Leiterplatte im Laufe der Zeit öfter

geändert worden ist. Die abgebildete Platine stellt den zur Zeit der Drucklegung gültigen Stand dar.

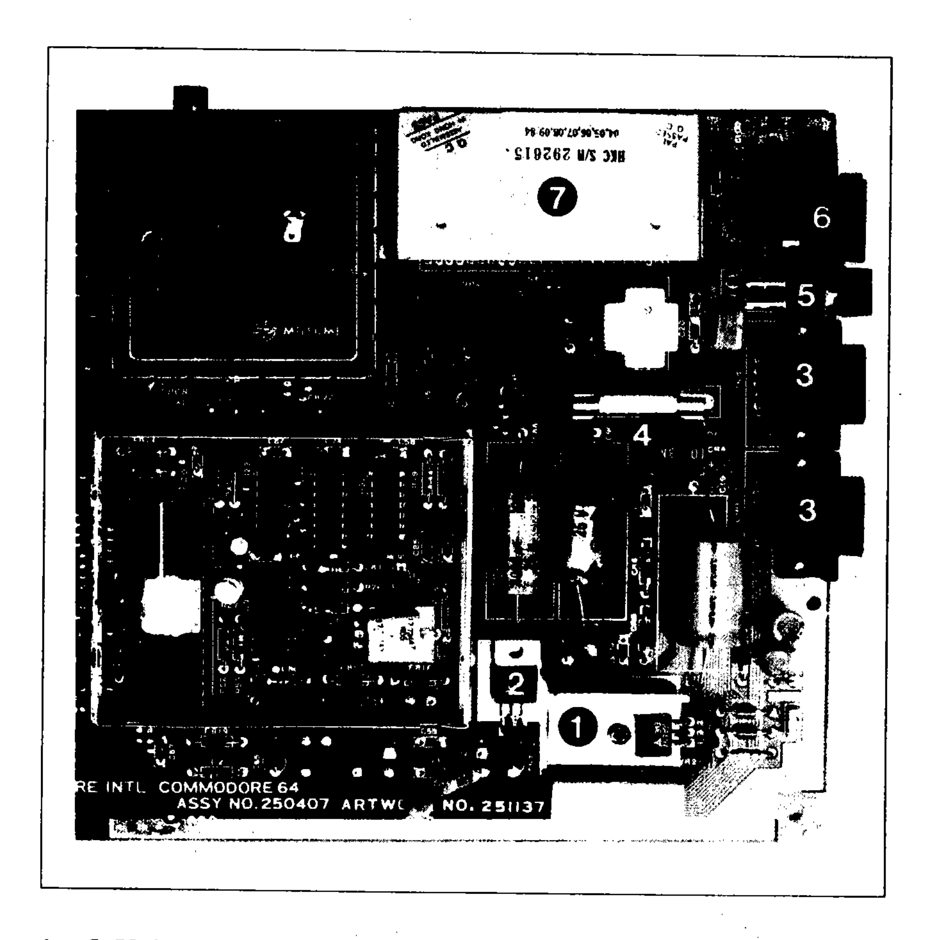
Ein Blick ins Innere



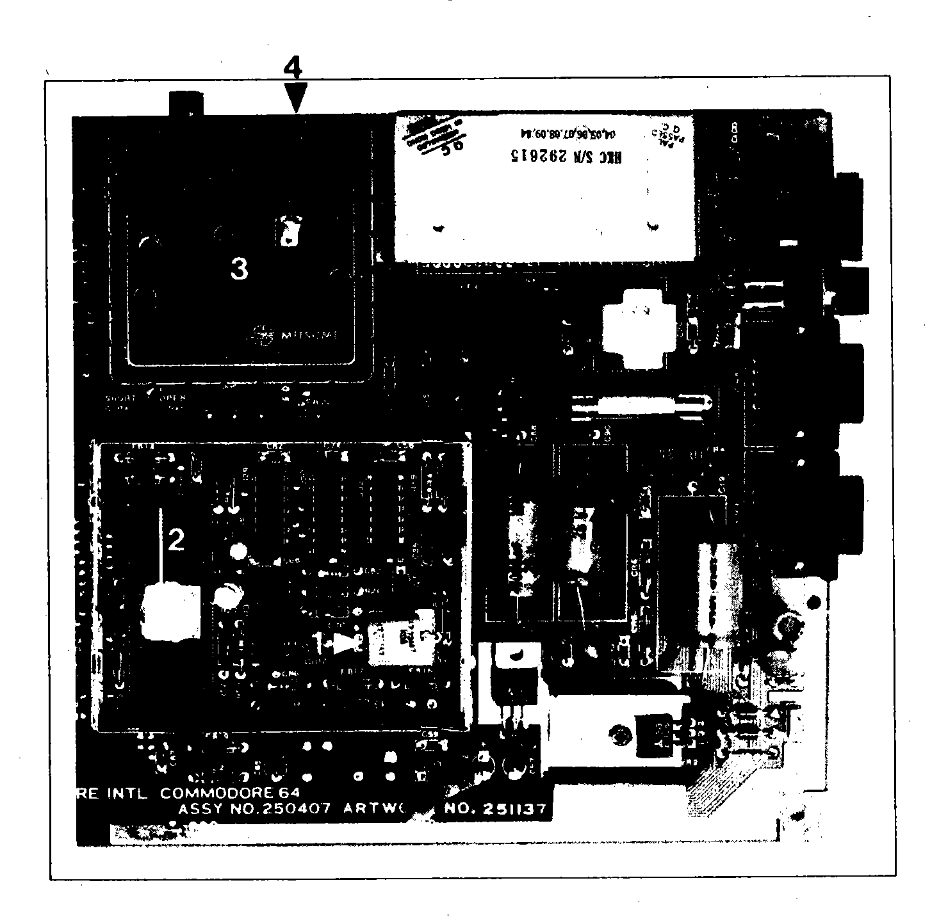
1. SID 6581
2. Character-ROM
3. Kernal-ROM
4. BASIC-ROM
5. CIA U2, User-Port, opt. RS-232, teilw. serieller IEC-Bus, Videohilfsadressen
6. CIA U1, Tastaturabfrage, Control-Ports Cass-Read
7. Anschluß für die Tastatur



1. Memory-Manager
2. Farb-RAM
3. Prozessor 6510
4. Transistorstufe zur Motorsteuerung der Datasette
5. 64K-RAM-Multiplexer



1. 5-Volt-Festspannungsregler
2. 12-Volt-Festspannungsregler
3. Control-Ports
4. Sicherung
5. Ein-/Aus-Schalter
6. Stromversorgung-Stecker
7. Expansion-Port



1. Quarz
2. VIC 6569
3. Modulator
4. Einstellregler für Kanalfrequenz des Modulators. Kann bei Bedarf verstellt werden, wenn ein starker Fernsehsender auf dem Kanal liegt.

Liste der verwendeten Halbleiter

Für die Spezialisten unter Ihnen hier noch eine Aufstellung der im CBM 64 verwendeten ICs mit Herstellerangaben. Damit haben Sie die Möglichkeit, Ihr Gerät selbst zu reparieren.

Bez.	Typenbez.	Hersteller
U1	6526 CIA	Commodore MOS
U2	6526 CIA	Commodore MOS
U3	2364A BASIC	Commodore MOS
U4	2364A KERNAL	Commodore MOS
U5	2332A CHARACTER	Commodore MOS
U6	2114L-3 COLOR RAM	Diverse Hersteller, z.B.
		OKI MSM 2114L-3
		FAIRCHILD 2114L-3
		HITACHI HM2114L-3
		MOS MPS2114L-30
		MOTOROLA MCM2114L-30
		NEC uPD2114L-1
U7	6510 MPU	Commodore MOS
U8	7406	Diverse Hersteller
U9	4164 RAM	Diverse Hersteller
U10		
U11	z.B. NEC	uPD4164-2
U12	MOSTEK	MK4164-10
U21		
U22		
U23		
U24		
U13	SN74LS257	Diverse Hersteller
U14	SN74LS278	Diverse Hersteller
U15	SN74LS139	Diverse Hersteller
U16	MC4066	Diverse Hersteller
U17	825100	Signetics, programmiert durch Commodore
U18	6581 SID	Commodore MOS
U19	6589 VIC	Commodore MOS
U20	556	Diverse Hersteller
U25	SN74LS257	Diverse Hersteller
U26	SN74LS373	Diverse Hersteller
U27	SN74LS08	Diverse Hersteller
U28	MC4066	Diverse Hersteller
U29	SN74LS74	Diverse Hersteller
U30	SN74LS193	Diverse Hersteller
U31	SN74LS629	Diverse Hersteller
U32	MC4044	Motorola
VR1	7812 12V Regler	Diverse Hersteller
VR2	7805 5V Regler	Diverse Hersteller

Die Stromversorgung

Obwohl die Stromversorgung zu den einfachen Schaltkreisen in einem Computer gehört, hat der Entwickler doch einige Tricks angewendet, um mit minimalem Aufwand einen größtmöglichen Effekt zu erzielen. Den Anschluß an das Lichtnetz übernimmt der Trafo. Dieser Trafo befindet sich zusammen mit einer Gleichrichterschaltung im Trafogehäuse und wird über einen 7-poligen DIN-Stecker an die mit CN7 bezeichnete Buchse angeschlossen. Im Trafo wird eine Wechselspannung von 9 Volt erzeugt, die auf die Pins 6 und 7 von CN7 geführt werden. Die Gleichrichterschaltung im Trafogehäuse erzeugt über eine zweite Trafowicklung eine stabilisierte Gleichspannung von 5 Volt. Diese 5 Volt liegen auf dem Pin 5 von CN7, die Masseleitung auf den Pins 1, 2 und 3. Die von den Buchsenkontakten kommenden Spannungen werden zur Beseitigung von Netzstörungen über die Spulen L2 und L4 und die Kondensatoren C20, C21, C98, C99 und C100 geführt und gefiltert.

Der mit SW1 bezeichnete doppelpolige Schalter ist der an der rechten Seite befindliche Einschalter. Die 9-Volt-Wechselspannung wird mit der Sicherung F1 (1 Ampere) abgesichert und steht am User-Port an den Kontakten 11 und 12 zur Verfügung. Diese Spannung können Sie nach Gleichrichtung und Siebung für externe Geräte verwenden; belasten Sie diese Stromquelle jedoch nur mit maximal 100 mA, die Sicherung wird es Ihnen danken. Apropos Sicherung:

Wenn sie defekt ist, leuchtet die LED am 64, auch eine angeschlossene Floppy macht beim Einschalten des Rechners einen RESET, auf dem Bildschirm ist aber nichts zu sehen. Vergewissern Sie sich aber zuerst, ob der Fernseher auf dem richtigen Kanal steht und das HF-Kabel angeschlossen ist. Wenn alles richtig erscheint, kontrollieren Sie die Sicherung. Ist diese durchgebrannt, dann ersetzen Sie sie durch eine Sicherung mit dem Wert 1.25 Ampere. Sollte auch die neue Sicherung ihren "Geist" aufgeben, liegt mit einiger Wahrscheinlichkeit ein Defekt vor.

Nach der Sicherung kommt eine Gleichrichterschaltung, welche 5 Volt stabilisiert, 9 Volt ungeregelt sowie 12 Volt stabilisiert zur Verfügung stellt.

Die Gleichrichterschaltung besteht aus dem Brückengleichrichter CR4 und den Dioden CN5 und CN6. Hinter dem Brückengleichrichter stehen die ungeregelten 9 Volt, die mit VR2, einem integrierten 5V-Festspannungsregler, auf 5 Volt stabilisiert werden. Über die Dioden CN5 und CN6 wird die Wechselspannung auf eine ungeregelte Gleichspannung von ca. 16 Volt gleichgerichtet, die mit dem Spannungsregler VR1 auf 12 Volt stabilisiert wird. Die aus dem Trafogehäuse kommenden 5 Volt sind mit einem eigenen Spannungsregler schon im Trafogehäuse stabilisiert. Dies hat den Vorteil, daß die erzeugte Verlustwärme nicht den Computer aufheizt; der erzeugt schon genug eigene Wärme. Diese Spannung übernimmt die Versorgung der meisten ICs in Ihrem C64 und liegt am Pin 2 des User-Ports CN2. Damit steht Ihnen für kleinere Projekte schon eine geeignete Spannung zur Verfügung. Aber auch diese Spannungsquelle sollten Sie nicht überlasten. Der Maximalstrom ist mit 100 mA angegeben, sicherlich für einige ICs ausreichend. Erfreulicherweise ist diese Spannung kurzzeitig kurzschlußfest. Dieser Kurzschlußfall ist sehr einfach feststellbar: In diesem Fall erscheint kein Bild auf dem angeschlossenen Fernseher und die Leuchtdiode leuchtet nicht, da auch sie von dieser Spannung versorgt wird.

Die im C64 erzeugte Spannung von 5 Volt trägt die Bezeichnung CAN+5. Diese Spannung versorgt den Video-Controller (weiterhin kurz als VIC bezeichnet), die Video-Ausgangsstufe und alle zur Takterzeugung benötigten ICs. Der VIC bekommt die 5 Volt direkt, zur Video-Ausgangsstufe wird die Spannung über die Spule L1 und die Kondensatoren C61, C63 und C64 gefiltert. Alle der Takterzeugung zugehörigen Bauteile bekommen die Spannung über L2, C65, C66 und C67 gesiebt zugeführt. Da die Datasette kein eigenes Netzteil hat, muß der Computer auch den benötigten "Saft" hierfür liefern. Die von der Datasette benötigten Spannungen sind 6 Volt für den Rekordermotor und 5 Volt für die eingebaute Elektronik. Der Antriebsmotor bekommt die Spannung über die Transistorschaltung Q1, Q2 und Q3 auf die Kontakte 3 und C des Kassetten-Port-Steckers CN3 geschaltet.

Wenn der Prozessor das Port-Bit 5 auf High legt, wird der Transistor Q2 durchgeschaltet. Damit ist die Zenerdiode CR2 kurzgeschlossen, der Transistor Q1 bekommt keine Basisvorspannung, Q1 und Q3 sperren. Der Rekordermotor stoppt. Wird das Port-Bit dagegen Low, dann ist der Transistor Q2 gesperrt. An der Basis von Q1 liegt die Zenerspannung von 7.5 Volt und steuert die Transistoren Q1 und Q3 an. Am Emitter von Q3 liegt die um die beiden Basis-Emitterspannungen der Transistoren (ca 1.5 Volt) reduzierte Zenerspannung, das ergibt ca. 6 Volt. Durch diese Stabilisierung der Motorschaltstufe wird eine konstante Drehzahl des Motors erreicht. Die Elektronik der Datasette wird über die Kontakte 2 und B des Steckers CN3 versorgt. Bleiben noch die 12 Volt. Diese Spannung wird für den VIC, den SID (Sound Interface Device) und die Audio-Ausgangsstufe mit dem Transistor Q8 benötigt.

Nicht direkt zur Stromversorgung gehörend ist die kleine Schaltung rund um das Gatter U27. Trotzdem soll sie hier erläutert werden, da sie ihre Signale aus dem Netzteil bekommt. Das Gatter U27 stellt eine UND-Verknüpfung dar. Der Eingang Pin 13 liegt fest an 5 Volt, der Eingang Pin 12 über den Widerstand R5 an den 9 Volt Wechselspannung. Am Pin 12 würde sich die Spannung also mit der Netzfrequenz von 50 Hertz ändern. Nun ist eine Spannung von 9 Volt für einen TTL-Eingang nicht sehr verträglich und eine negative Spannung von -9 Volt sollte an einem solchen Eingang unbedingt vermieden werden, um das IC nicht zu zerstören. Um die Eingangsspannung zu begrenzen, ist die Zenerdiode CR1 an den Eingang geschaltet. Wenn die Wechselspannung über +2,7 Volt steigt, so wird sie von der Zenerdiode auf diesen Wert begrenzt. Damit ist ein logisches High-Signal gegeben. Die negative Spannung wird von der Zenerdiode auf -0.7 Volt begrenzt, ein Wert, den der TTL-Eingang noch gut verkraftet und der als logisches Low-Signal erkannt wird. Die Spannung schwankt also im Rhythmus der Netzfrequenz am Pin 12 des U27 zwischen Low und High. Damit ändert sich der Ausgang im selben Takt.

Der Widerstand R37 stellt eine Mitkopplung dar, er beschleunigt die Anstiegs- und Abfallzeiten, um saubere Rechteckimpulse für die weitere Verwendung zur Verfügung zu stellen.

Woraus besteht nun die weitere Verwendung? Im Schaltplan kann man erkennen, daß diese 50 Hertz an die ICs U1 und U2, die beiden CIAs, gehen. Auf die CIAs wird im weiteren Verlauf der Schaltplanbeschreibung noch näher eingegangen. Jetzt nur so viel: Die Netzfrequenz ist das am einfachsten zu erzeugende frequenzkonstante Signal. Darum eignet es sich besonders für Anwendungen, in denen Zeiten gemessen werden sollen. Das ist auch Aufgabe des Signals in den CIAs. Diese enthalten sogenannte Echtzeituhren, die ihren Takt von der Netzfrequenz beziehen.

Die Takterzeugung

Für ein ordnungsgemäßes Funktionieren eines Computers ist eine stabile und störungsfreie Stromversorgung sehr wichtig. Die Konstanz und Stabilität der Taktsignale ist für die Funktion aber sicher genauso maßgebend. Dieser Takterzeugung wollen wir uns jetzt zuwenden.

Wenn Sie auf die Leiterplatte des CBM 64 schauen und mit dem Schaltplan auf IC-Suche gehen, so werden Sie vermutlich das eine oder andere IC nicht auf den ersten Blick finden, genauso wenig wie die für die Taktversorgung zuständigen ICs. Diese befinden sich zusammen mit dem VIC (Video Interface Controller) in dem Blechkasten in der Mitte der Platine (nicht der Kasten mit dem Fernsehanschluß, das ist der UHF-Modulator). Dieses Blechgehäuse schirmt die bei der Takterzeugung entstehende hochfrequente Störstrahlung ab. Bei Rechnern ohne ausreichende Abschirmung kann man beobachten, daß alle im näheren Umkreis befindlichen Radios nur Pfeif- und Zischlaute von sich geben. Schlimmer noch, auch Fernsehgeräte werden von solchen Störstrahlungen beeinflußt. Wenn der 64 nicht über ausreichende Entstörmaßnahmen verfügen würde, wäre der Betrieb mit einem Fernseher, wenn auch nicht unmöglich, so doch sehr gestört.

Die alles bestimmende Taktfrequenz wird vom Quarz Y1 erzeugt. Doch vorab noch eine Erläuterung. Alle jetzt folgenden Angaben beziehen sich auf ein für den deutschen Markt produziertes Gerät mit PAL-Ausgang. Der Quarz Y1 schwingt mit ei-

ner Frequenz von 17,734472 MHz. Er ist über C70 an das IC U31 angeschlossen. Das IC U31, ein TTL-IC mit der Bezeichnung 74LS629, enthält 2 unabhängige VCOs. Ein VCO ist ein spannungsgesteuerter Oszillator. Durch eine am Steuereingang angelegte Gleichspannung kann die Frequenz in einem bestimmten Bereich verändert werden. Dieser Steuereingang ist für den VCO 1 der Pin 1. Das Poti R27 an diesem Eingang erlaubt eine - wenn auch geringfügige - Änderung der Ausgangsfrequenz. Da auch Quarze eine gewisse Toleranz haben, läßt sich die Soll-Frequenz mit dem Poti genau einstellen. Der Ausgang des VCO 1 ist der Pin 10. Die hier anliegende Frequenz wird direkt als Signal 0COLOR an den VIC geführt. Gleichzeitig gelangt das Signal an das IC U30. Dieses IC, ein 74LS193, ist als Frequenzteiler geschaltet. Dieser Teiler hat ein einstellbares Teilerverhältnis. In Abhängigkeit der Pegel an den Pins 1, 9, 10 und 15 läßt sich jedes Teilerverhältnis zwischen 1:1 und 15:1 einstellen. In unserem Fall ist das Teilerverhältnis auf 9:1 eingestellt. Die 17,734 MHz werden also durch 9 geteilt. Damit steht am Ausgang Pin 6 eine Frequenz von 1,9704 MHz zur Verfügung. Diese Frequenz wird auf den Pin 11 des IC U29 geführt. U29 enthält 2 Flipflops. Mit jeder positiven Flanke des Clock-Signals an Pin 11 wird die am Dateneingang Pin 12 des Flipflops 1 liegende Information auf den Q-Ausgang Pin 9 weitergegeben. Der Ausgang -Q (Pin 8) hat dann auch die Eingangsinformation, nur mit invertierter Polarität. In der vorliegenden Beschaltung liefert die durch 9 geteilte Quarzfrequenz das Clocksignal für FF1. Der Dateneingang ist mit dem Ausgang -Q verbunden. Wenn dieser -Q-Ausgang High ist, wird das High-Signal mit der nächsten positiven Flanke an Pin 11 auf den Q-Ausgang gegeben. Gleichzeitig wird der -Q-Ausgang Low. Mit der nächsten positiven Taktflanke wird das Low an Q gelegt, -Q hat jetzt wieder ein High und so weiter.

Mit jedem zweiten Taktimpuls wechseln also die Ausgänge ihren Zustand. Das kommt einer Frequenzteilung durch den Faktor 2 gleich, am Ausgang erscheint eine Frequenz von 985,248 KHz. Das ist die Taktfrequenz des Prozessors. Dieses Signal wird aber nicht direkt als Takt verwendet, die ganze Sache ist etwas komplizierter. Das Signal Dot Clock mit der Frequenz 7,88198 MHz läßt sich durch Frequenzteilung nicht aus der Quarzfrequenz

ableiten. Darum muß ein anderer Weg beschritten werden, die Frequenzsynthese mit einer PLL-Schaltung. PLL bedeutet "Phase Locked Loop", übersetzt etwa "phasengeregelte Schleife". Der PLL im 64 ist mit den ICs U32, U31 und dem VIC aufgebaut. Wichtigster Bestandteil eines PLL ist ein Phasencomparator mit zwei Eingängen. Dieser Phasencomparator liefert an seinem Ausgang eine Gleichspannung, die proportional der Phasenlage der beiden Signale ist. Diese Funktion ist mit dem IC U32 und dem Transistor Q7 aufgebaut. Im Detail funktioniert die Sache so:

Am Eingang Pin 1 des U32 liegt eine Frequenz von 985 KHz, geliefert vom Ausgang des Flipflop U29. Am zweiten Eingang des PLL Pin 3 liegt das Signal 0o, das vom VIC gelieferte Taktsignal für den Prozessor, mit noch unbestimmter Frequenz. Dieses Signal 0o vom VIC stellt das durch 8 geteilte Ausgangssignal des VCO 2 im U31 dar. Die Frequenzteilung durch 8 findet direkt im VIC statt. Die Frequenz des VCO 2 wird nicht durch einen Quarz, sondern durch einen Kondensator, den C86, bestimmt. Die Steuerspannung des VCO 2 wird jetzt durch den Ausgang des Phasencomparators U32 geliefert. Wenn die Steuerspannung des VCO 2 ca. 3 Volt beträgt, schwingt er auf einer Frequenz von 7,88198 MHz. Wenn wir jetzt den Fall annehmen, daß die Frequenz des Flipflops U29 höher als die Frequenz 0o ist, der VCO 2 also beispielsweise nur mit 7.7 MHz schwingt, dann liefert der Ausgang Pin 8 des Phasencomparators eine Spannung kleiner 3 Volt, die den VCO mit einer höheren Frequenz schwingen läßt. Damit erhöht sich auch die Frequenz am Pin 3 des Phasencomparators, sie nähert sich der Referenzfrequenz am Pin 1, die Steuerspannung nähert sich den 3 Volt. Wenn die Frequenzen an Pin 1 und Pin 3 gleich sind, wird der VCO noch so lange geregelt, bis die Signale nicht nur frequenz-, sondern auch phasengleich sind. Derselbe Vorgang läuft ab, wenn der VCO mit zu hoher Frequenz schwingt. Dann wird die Steuerspannung größer 3 Volt. Jetzt schwingt der VCO langsamer und die Steuerspannung nimmt ab, bis die Signale frequenz- und phasengekoppelt sind. Dann liegt das Signal "Dot Clock" richtig an. Die geschilderten Regelvorgänge brauchen nur kurze Zeit. Nach spätestens 100 Millisekunden stehen alle Frequenzen zur Verfügung. Zum Abschluß noch eine kurze Schilderung der Funktion des FF2 und der Abläufe in einem 64 mit

NTSC-Farbausgang. In diesen für den amerikanischen Markt produzierten Geräten ist zum einen ein 14.31818 MHz-Quarz eingebaut. Des weiteren ist ein anderer VIC-Chip, ein 6567, in der NTSC-Version eingesetzt. Bei der PAL-Version ist dies ein 6569.

Als drittes Merkmal ist die Drahtbrücke zwischen den Punkten E1 und E2 oder E3 anders gelegt. Bei PAL-Geräten ist diese Brücke zwischen E1 und E2 geschaltet. Damit liegen die Pins 1 und 10 des Teilers U30 an +5 Volt. Auch der Pin 4 des IC U29 liegt an High. Dieser Pin 4 ist der sogenannte Preset-Eingang an FF2. Clock-, Daten- und Clear-Eingang dieses FFs sind an Masse gelegt. Ein Low-Signal am Clear-Eingang versetzt das Flipflop in einen definierten Zustand. Unabhängig von den anderen Eingangssignalen wird der Q-Ausgang Low, -Q dagegen High. Wie bei so vielen anderen Gelegenheiten gibt es aber eine wichtige Einschränkung zu dem zuvor Gesagten: Um diesen Zustand zu erhalten, muß der Preset-Eingang auf High-Pegel liegen.

Diese Bedingung ist bei einem PAL-Gerät über die Drahtbrücke erfüllt. Die Eingänge 1, 9, 10 und 15 des Zählers U30 bestimmen binär codiert den Startwert des Zählers. Da der Zähler immer bis 16 zählt, kann man mit dem Startwert das Teilerverhältnis einstellen. Er beginnt dann nicht bei 0, sondern mit dem programmierten Wert. Der Eingang A stellt das niederwertige Bit dar, Eingang D das höchstwertige Bit. An diesen Eingängen liegt dezimal ausgedrückt eine 7. Der Zähler zählt bis 16 weiter und beginnt dann wieder bei 7. Für diesen Durchlauf benötigt er 9 Zählimpulse, er teilt also durch 9. Damit stellt das FF2 nichts anderes als einen einfachen "Inverter" dar. Wenn der Eingang High ist, so ist der Ausgang Low und umgekehrt, die normale Inverter-Funktion. Bei NTSC liegt der Preset-Eingang des U29 auf Low. Laut Datenblatt haben jetzt sowohl Q- wie auch -Q-Ausgang High-Pegel, eigentlich ein ungewöhnlicher Zustand, der das IC aber nicht beschädigt. Jetzt ist das Teilerverhältnis von U30 7:1, mit dem nachfolgenden Flipflop 14:1, und die Taktfrequenz des Prozessors beträgt damit 1.0227 MHz, ist also geringfügig höher als die PAL-Arbeitsfrequenz.

Der Prozessor

Wie schon erwähnt ist der Prozessor des C64 der 6510. Dieser neue Prozessor unterscheidet sich von dem bekannten 6502 in der Hauptsache durch einen im Prozessorchip integrierten Port. Dieser Port verfügt über 6 programmierbare IO-Leitungen (IO = Abkürzung für Input Output; Leitungen, die wahlweise als Eingänge oder Ausgänge geschaltet sind).

Die Zahl 6 ist im Zusammenhang mit 8-Bit-Prozessoren sicher etwas ungewöhnlich. Bei dem zur Verfügung stehenden 40-poligen Gehäuse waren aber nicht mehr Leitungen frei, um einen vollen 8-Bit-Port zu realisieren. Die 40 Pins des 6510 sind wie folgt belegt:

Pin	Bez.	Funktion
1	0IN	Eingang, Systemtakt vom VIC Pin 17
2	RDY	Eingang, Ready von U27 Pin 3
3	-IRQ	Eingang, Interrupt Request
4	-NMI	Eingang, Non Maskable Interrupt
5	AEC	Eingang, Adress Enable Control
6	VCC	Betriebsspannung +5V
7	A0	Ausgang, Adreß-Bit 0
bis		
20	A13	Ausgang, Adreß-Bit 13
21	GND	Betriebsspannung Masse
22	A14	Ausgang, Adreß-Bit 14
23	A15	Ausgang, Adreß-Bit 15
24	PB5	Ein-Ausgang, Port-Bit 5
bis		
29	PB0	Ein-Ausgang, Port-Bit 0
30	D7	Ein-Ausgang, Daten-Bit 7
bis		
37	D0	Ein-Ausgang, Daten-Bit 0
38	R/-W	Ausgang, Read/-Write
39	02	Taktausgang Phase Two, im folgenden 02 genannt
40	-RES	Eingang, RESET

Wie viele andere Prozessoren hat also auch der 6510 einen 8-Bit-Daten- und einen 16-Bit-Adreßbus. Somit kann der 6510 einen Speicherbereich von 64 K direkt adressieren.

MOS 6510

Signal	Pin	Pin	Signal
Φ0 IN	1	40	$\overline{RES}$
RDY	2	39	PH2 OUT
$\overline{IRQ}$	3	38	R / $\overline{W}$
$\overline{NMI}$	4	37	DB0
AEC	5	36	DB1
VCC	6	35	DB2
A0	7	34	DB3
A1	8	33	DB4
A2	9	32	DB5
A3	10	31	DB6
A4	11	30	DB7
A5	12	29	P0
A6	13	28	P1
A7	14	27	P2
A8	15	26	P3
A9	16	25	P4
A10	17	24	P5
A11	18	23	A15
A12	19	22	A14
A13	20	21	GND

Abb. A.1: Die 6510

MODUL – STECKPLATZ :

22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Z Y X W V U T S R P N M L K J H F E D C B A

Pin	Signal
1	GND
2	+ 5V
3	+ 5V
4	IRQ
5	CR/W
6	DOT CLOCK
7	I/0 1
8	GAME
9	EXROM
10	I/0 2
11	ROML
12	BA
13	DMA
14	D 7
15	D 6
16	D 5
17	D 4
18	D 3
19	D 2
20	D 1
21	D 0
22	GND
A	GND
B	ROMH
C	RESET
D	NMI
E	Φ 2
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND

Abb. A.2: Expansion-Port

Die Signale 0IN und 02 sind die Taktsignale des Systems, sozusagen der Herzschlag des Rechners. Das Signal 0IN wird vom VIC erzeugt und hat eine Frequenz von ungefähr 985 KHz. Aus diesem Signal wird im Prozessor das Signal 02 erzeugt. 02 ist für das Zusammmenspiel von Prozessor und Peripherie sehr wichtig, es stellt den Bezugstakt für alle Operationen des Prozessors dar.

Das Signal -RES wird benutzt, um den Prozessor und andere ICs in einen definierten Anfangszustand zu versetzen. Dieser RESET findet im Einschalt-Moment statt. Schauen wir uns diesen Einschalt-Moment einmal etwas näher an. Das -RES-Signal wird vom IC U20 erzeugt. Dies IC, ein NE556, enthält 2 identische Timer-Baustufen. Mit diesen Timern kann man durch einfache externe Beschaltung Oszillator- oder Impulsgeberbaustufen aufbauen. In unserem Fall ist das IC als Impulsgeber geschaltet. Mit dem Anlegen der Betriebsspannung wird der Kondensator C105 über den Widerstand R50 aufgeladen. Gleichzeitig wird der Kondensator C24 über den Widerstand R34 aufgeladen. Wenn nun nach einiger Zeit (einigen 10 Millisekunden) die Spannung am C105 den Wert von 1.6 Volt (1/3 der Betriebsspannung) übersteigt, wird der eigentliche Impuls gestartet. Der Kondensator C24 wird über den Anschluß 13 schlagartig entladen. Gleichzeitig wird der Pin 9, der Ausgang des Timers, auf 5 Volt gelegt. Danach wird C24 über den Widerstand R34 wieder aufgeladen. Aber jetzt wird die Spannung durch den Eingang Pin 12 überwacht. In dem Moment, wo die Spannung 2/3 der Betriebsspannung (ca. 3.3 Volt) übersteigt, wird der Ausgang wieder Low. Dieser Zeitpunkt ist nach etwa .5 Sekunden erreicht. Der am Ausgang Pin 9 des Timers befindliche Inverter macht aus diesem positiven Impuls einen negativen. An seinem Ausgang steht das eigentliche -Res-Signal zur Verfügung. Im Moment des Wechsels von High nach Low startet der Prozessor seine Arbeit. Als erstes holt er von den Adressen $FFFC und $FFFD (genannt RESET-Vektor) die Adresse des nächsten zu verarbeitenden Befehls. Auf dieser Adresse beginnt nun das eigentliche Betriebssystem.

Der Pin mit der Bezeichnung R/-W signalisiert, ob der Prozessor einen Lese- oder einen Schreibzugriff vornimmt. Wenn diese Leitung High ist, liest der Prozessor Daten aus RAM, ROM oder

Interfacechips. Bei einem Low auf dieser Leitung schreibt der Prozessor, d.h., er speichert Daten im jeweils adressierten Baustein. Dieses Schreiben ist natürlich nur dann sinnvoll, wenn der adressierte Baustein diese Daten auch speichern kann. Auf ein ROM zu schreiben ist wenig sinnvoll, da die Daten des ROM schon bei der Herstellung festgelegt werden und nicht veränderbar sind.

Der Pin mit der Bezeichnung -NMI (Non Maskable Interupt = nicht maskierbare oder ausblendbare Unterbrechnung) gestattet die Unterbrechung eines gerade laufenden Programms. Nicht maskierbar bedeutet, daß der Interrupt immer zugelassen ist. Er ist durch Software nicht auszuschließen. Wann immer dieser Anschluß nach Masse gezogen wird, wird mit der Beendigung des gerade abgehandelten Maschinensprachebefehls das laufende Programm verlassen. Der Prozessor holt vom NMI-Vektor ($FFFA und $FFFB) die Adresse der Interrupt-Routine, und verzweigt auf diese. Der NMI kann im CBM 64 durch drei verschiedene Ereignisse ausgelöst werden:

Der erste Fall ist das Drücken der <Restore>-Taste. Wird diese Taste gedrückt, dann erzeugt der zweite Timer des U20 einen geeigneten Impuls. Das Drücken der Taste entlädt den Kondensator C38 schlagartig. Über den Widerstand R35 wird der Kondensator wieder aufgeladen, auch wenn die <Restore>-Taste noch gedrückt ist. Sobald die Spannung am Pin 6 des U20 1.6 Volt übersteigt, wird der eigentliche NMI-Impuls gestartet. Der Ausgang des Timers Pin 5 wird High, am Ausgang des Inverters U6 erscheint ein Low-Pegel, der Kondensator C23 wird über den Pin 1 von U20 entladen und beginnt sich über R33 wieder aufzuladen.

Nach ca. 18 Microsekunden ist der C23 auf 2/3 der Betriebsspannung aufgeladen und der Ausgang Pin 5 wird wieder Low, der -NMI-Eingang des Prozessors ist wieder High. Der zweite Fall wird durch die CIA U2 erzeugt. Der Pin 21 dieses ICs kann beim Eintreffen bestimmter Ereignisse einen Low-Pegel annehmen. Die Erzeugung dieses -NMI wird im Abschnitt über die CIAs behandelt.

Der dritte Fall ist das Kurzschließen des Anschlusses D der Cartridge-Expansion. Hier können externe Bausteine einen Interrupt auslösen.

Dem -NMI ähnlich ist der -IRQ (Interrupt ReQuest). Als wesentlicher Unterschied zum -NMI ist zum einen der Interrupt-Vektor des -IRQ zu sehen. Dieser Vektor liegt auf den Adressen $FFFE und $FFFF. Des weiteren ist dieser Interrupt softwaremäßig ausschaltbar.

Wenn im Prozessor-Status-Register das I-Flag (Bit 2) gesetzt ist, werden alle auftretenden Interrupts ignoriert. Ein weiterer Unterschied zum -NMI ist die Tatsache, daß der -IRQ nicht flankengesteuert ist. Der Interrupt muß also mindestens so lange anliegen bis der Prozessor diesen Anschluß prüft. Erzeugt wird der -IRQ auch wieder auf drei verschiedene Arten:

Die CIA U1 erzeugt an seinem Pin 21 genau wie die CIA U2 einen Low-Pegel beim Erreichen bestimmter programmierbarer Zustände. Dieser Low-Pegel erzeugt einen -IRQ am Prozessor.

Die zweite Möglichkeit der Interrupt-Erzeugung ist der VIC. Am Pin 8 des VIC erscheint genau wie bei den CIAs beim Erreichen bestimmter, vorher durch Programmierung festgelegter Ereignisse ein Low-Pegel und damit der -IRQ.

Die dritte Möglichkeit der -IRQ-Erzeugung besteht im Kurzschließen des Anschlusses 4 des Cartridge-Expansion-Steckers (CN6). Somit haben auch externe Schaltungen die Möglichkeit der -IRQ-Generierung. Der RDY-Pin zeigt dem Prozessor, ob die auf dem Datenbus liegenden Informationen gültig sind oder nicht.

Immer wenn dieser Pin Low ist, wird dem Prozessor signalisiert, daß er die Daten noch nicht übernehmen kann. Der Prozessor geht dann in einen sogenannten Wartezustand und stellt seine Aktivitäten ein. Er prüft nur mit jedem Taktimpuls, ob der RDY-Pin wieder High ist.

In älteren Prozessorsystemen wurde diese Möglichkeit genutzt, um langsame Speicher- und Peripheriebausteine am Prozessor anzuschließen. Im CBM 64 wird dieses Signal vom VIC genutzt. Normalerweise geschieht der Zugriff des VIC auf das RAM nur in den vom Prozessor nicht genutzten Taktlücken (02 = Low).

Bei bestimmten Operationen des VIC, z.B. Darstellung der Sprites, benötigt der VIC mehr Zeit, als in den Taktlücken zur Verfügung steht. Dann erzeugt der VIC am Anschluß BA (Bus Available) ein Low, welches über das AND-Gatter U27 an den RDY-Eingang des Prozessors geführt wird, worauf der Prozessor den Bus dem VIC für die benötigte Zeit zur Verfügung stellt. AEC ist ebenfalls ein in der Grundkonfiguration vom VIC erzeugtes Signal.

Immer wenn der VIC den Bus belegt, wird dieser Anschluß 0. Dieses Low-Signal wird an den AEC-Pin des Prozessors geführt und bewirkt, daß der Prozessor seine Busleitungen in einen hochohmigen, den sogenannten Tri-State versetzt. In der Praxis wirkt das, als ob der Prozessor gar nicht in seinem IC-Sockel säße. Solange AEC Low ist, bleibt dieser Zustand erhalten und andere ICs, z.B ein externer Prozessor oder der VIC, können den Systembus belegen.

Der im Prozessor integrierte Port belegt die Pins 24 bis 29. Im CBM 64 werden verschiedene Aufgaben von diesem Port übernommen. Im Einzelnen sind das die folgenden Funktionen:

Das Port-Bit 0 trägt die Bezeichnung -LOWRAM. Dieses Bit schaltet im Adreßbereich $A000 bis $BFFF zwischen RAM und ROM, d.h. bei Low-Pegel ist in diesem Adreßbereich RAM eingeschaltet. Port-Bit 1 mit der Bezeichnung -HIRAM übernimmt dieselbe Funktion im Adreßbereich von $E000 bis $FFFF. Port-Bit 2 mit der Bezeichnung -CHAREN selektiert, wenn es einen Low-Pegel hat, das Character-ROM.

Character-ROM und der sogenannte IO-Bereich belegen denselben Adreßbereich von $D000 bis $DFFF. Über -CHAREN wird also entschieden, ob das Character-ROM oder die den gleichen

Adreßbereich benutzenden IO- oder Peripherie-Bausteine VIC, SID oder CIAs selektiert sind.

Die drei verbleibenden Bits sind für den Betrieb der Datasette reserviert. Die Schreibdaten für die Datasette werden vom Port-Bit 3 geliefert. Dieser Prozessor-Pin wird direkt auf die Anschlüsse E und 5 des Kassetten-Ports geführt. Port-Bit 4 (Cass Sense) überprüft, ob an der Datasette die <Play>-Taste gedrückt ist. Dieses Bit liegt direkt an den Anschlüssen F und 6 des Kassetten-Ports. Die Motorsteuerung des Rekorders wird von Bit 5 übernommen.

Adreßdecodierung

Da der 6510 nur einen Adreßraum von 64 K verwalten kann, dieser aber schon von den 64-K-RAM belegt wird, muß eine zusätzliche Logik die Verwaltung der sich teilweise überlappenden Speicherbereiche übernehmen. Diese Verwaltung ist in der Hauptsache in einem speziellen IC integriert, dem sogenannten Adreß-Manager. Im Schaltplan trägt dieses IC, ein FPLA (Field Programmable Logic Array), die Bezeichnung U17. Erst durch die Programmierung hat dieses IC seine besonderen Logikeigenschaften erhalten und ersetzt eine große Anzahl verschiedener Gatter, die nötig wären, wollte man die Funktion des AM mit herkömmlichen ICs nachbilden. Die Pin-Belegung dieses 28-poligen ICs sieht folgendermaßen aus:

Pin	Bez.	Funktion
1	FE	Nicht benutzt
2	I7	Eingang, A13 vom 6510 Pin 20
3	I6	Eingang, A14 vom 6510 Pin 22
4	I5	Eingang, A15 vom 6510 Pin 23
5	I4	Eingang, -VA14 vom CIA 2 Port A Bit 0 Pin 2
6	I3	Eingang, -CHAREN vom 6510-Port Bit 2 Pin 27
7	I2	Eingang, -HIRAM vom 6510-Port Bit 1 Pin 28
8	I1	Eingang, -LOWRAM vom 6510-Port Bit 0 Pin 29
9	I0	Eingang, -CAS vom VIC Pin 19
10	F7	Ausgang, -ROMH zum Expansion-Slot Pin B
11	F6	Ausgang, -ROML zum Expansion-Slot Pin 11
12	F5	Ausgang, -I/O zum Decoder U15 Pin 1
13	F4	Ausgang, GR/-W zum Farb-RAM U6 Pin 10
14	GND	Betriebsspannung Masse

Pin	Bez.	Funktion
15	F3	Ausgang, -CHAROM zum Character-ROM U5 Pin 20
16	F2	Ausgang, -KERNAL zum Kernal-ROM U4 Pin 20
17	F1	Ausgang, -BASIC zum BASIC-ROM U3 Pin 20
18	F0	Ausgang, -CASRAM zu den RAMs Pin 15
19	-OE	Eingang, Output Enable an Masse
20	I15	Eingang, -VA12 vom VIC Pin 28
21	I14	Eingang, -VA13 vom VIC Pin 29
22	I13	Eingang, -GAME vom Expansion Slot Pin 8
23	I12	Eingang, -EXROM vom Expansion Slot Pin 9
24	I11	Eingang, R/-W vom 6510 Pin 38
25	I10	Eingang, -AEC vom VIC Pin 16
26	I9	Eingang, BA vom VIC Pin 12
27	I8	Eingang, A12 vom 6510 Pin 19
28	Vcc	Betriebsspannung +5 V

Was bewirken jetzt die verschiedenen Eingangssignale an den Ausgängen des AM? Bei 16 Eingangsleitungen sind ja immerhin 65536 verschiedene Eingangskombinationen möglich. Da der AM jedoch nur 8 Ausgänge besitzt, ist schon ersichtlich, daß jeweils mehrere Eingangskombinationen eine bestimmte Ausgangskombination bewirken.

Aber auch unter den 256 möglichen Ausgangskombinationen sind nur wenige für den Computer wirklich sinnvoll. Übrigens, wenn jede mögliche Eingangskombination und die dazugehörige Ausgangskombination eine Zeile einer Seite belegen würden, dann hätte eine vollständige Liste bei dem von uns verwendeten Druckformat immerhin einen Umfang von 1093 Seiten.

Der Video-Controller 6569

Die beiden wichtigsten Peripheriegeräte eines Computers sind Eingabe- und Ausgabe-Einheiten, da sie die Möglichkeit schaffen, mit dem Computer in Verbindung zu treten. Die Ausgabe-Einheit des C64 ist in der Regel der Fernseher oder ein Monitor. Der VIC stellt im C64 alle für den Betrieb eines Fernsehers oder Monitors benötigten Signale zur Verfügung. Dies sind die Synchronisations- und Helligkeits-Impulse und die für Farbdarstellung benötigten Farbwerte.

Zusätzlich übernimmt der VIC aber noch andere Aufgaben. So erzeugt er den von der CPU benötigten Takt, übernimmt den bei

den verwendeten dynamischen RAMs notwendigen "Refresh" und liefert Steuersignale für den Betrieb der dynamischen RAMs. Diese Funktionen sind alle in einem 40-poligen Gehäuse untergebracht. Die Belegung der Pins ist in der folgenden Tabelle dargestellt.

Pin	Bez.	Funktion
1	D6	Prozessor-Datenbus
bis		
7	D0	Prozessor-Datenbus
8	-IRQ	Ausgang, Interrupt Request
9	-LP	Eingang, Light Pen
10	-CS	Eingang, Chip Select
11	R/-W	Read/-Write
12	BA	Bus Available
13	VDD	Betriebsspannung +12 Volt
14	COLOR	Ausgang, Farbinformation
15	SYNC	Ausgang, Zeilen- und Bildsynchronisations-Impulse
16	AEC	Ausgang, Adress Enable Control
17	0OUT	Ausgang, Systemtakt
18	-RAS	Ausgang, Row Adress Select
19	-CAS	Ausgang, Colum Adress Select
20	GND	Betriebsspannung Masse
21	0COLOR	Eingang, Farbfrequenz
22	0IN	Eingang, Dot-Frequenz
23	A11	Prozessor-Adreßbus
24	A0/A8	gemultiplexter (Video-) RAM-Adreßbus
bis		
29	A5/A13	gemultiplexter (Video-) RAM-Adreßbus
30	A6	(Video-) RAM-Adreßbus
31	A7	(Video-) RAM-Adreßbus
32	A8	Prozessor-Adreßbus
bis		
34	A10	Prozessor-Adreßbus
35	D11	Datenbus Farb-RAM
bis		
38	D8	Datenbus Farb-RAM
39	D7	Prozessor-Datenbus
40	VCC	Betriebsspannung +5 Volt

Wenn Sie sich die verschiedenen Pin-Bezeichnungen am VIC anschauen, dann treffen Sie auf einige bekannte Bezeichnungen. So sind BA, AEC, 02, und R/-W schon beim Prozessor erläutert worden. Völlig neu sind z.B. die Signale -CS, -RAS, -CAS und die Datenleitungen D8 - D11. Auch der gemultiplexte Adreßbus ist neu hinzugekommen, da am Prozessor ja alle Adreß-Signale

getrennt an einzelnen Pins zur Verfügung standen. Doch kommen wir zuerst zu den verschiedenen Taktsignalen.

Das den ganzen Zeitablauf im Rechner bestimmende Signal ist der Dot-Clock. Dieses Signal hat in Ihrem C64 eine Frequenz von ca. 7.85 MHz. Im VIC befindet sich eine Stufe, die diese Frequenz durch 8 teilt. Damit erhalten wir eine neue Frequenz von ca. 980 KHz. Diese Frequenz steht am Pin 17 als Systemtakt 0Out zur Verfügung.

Aus dem Dot-Clock werden weiterhin die Signale zur Synchronisation des Bildes auf dem Fernseher gewonnen. Der Dot-Clock selbst bestimmt die Zeit, mit der die einzelnen Punkte, aus denen alle Zeichen dargestellt werden, auf dem Bildschirm erscheinen. Die Frequenz des Signals 0COLOR beträgt in Ihrem C64 17.734472 MHz. Das ist die Frequenz, mit der der Quarz Y1 schwingt. Sie wird zur Erzeugung der Farbinformation benötigt.

Diese Frequenzen beziehen sich alle auf den Normalfall, d.h., der Rechner ist für den Betrieb mit einem PAL-System-Fernseher ausgestattet. Immer wenn der Prozessor auf die Register des VIC zugreifen will, muß der VIC adressiert werden. Dazu muß als wichtigstes die Leitung mit der Bezeichnung -CS auf Low gehen. Erst dann kann der Prozessor über die auf dem Adreßbus liegende Adresse das gewünschte Register ansprechen. Wie wird nun aber die Leitung -CS Low?

Da der VIC im sogenannten IO-Bereich ($D000 bis $DFFF) die Adressen von $D000 bis $D3FF belegt, erzeugt der AM bei einem Zugriff auf diesen Adreßbereich einen Low-Pegel an seinem Pin 12 (-I/O-Signal). Dieser Low-Pegel gelangt an den Decoder U15 Pin 1. Damit ist der Decoder freigegeben, und in Abhängigkeit von den Adreßleitungen A10 und A11 an den Pin 2 und 3 wird der entsprechende Ausgang des Decoders Low. Wenn man die Basisadresse und Endadresse des VIC einmal binär darstellt, so erhält man das folgende Bit-Muster:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0=$D000
1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1=$D3FF

Man sieht sofort, das die Adreß-Bits A10 und A11 in diesem Adreßbereich Low bleiben. Damit ist der Ausgang Y0 des Decoders auf Low, der VIC ist adressiert.

Erst bei der nächsten Adresse $D400 wird A10 High. Damit wird Y0 High, Y1 des Decoders wird Low und nun ist der SID adressiert. Der VIC kann nur einen Adreßraum von 16 K adressieren, er hat nur die Adreß-Bits A0 bis A13. Außerdem liegen die Adreßleitungen nicht wie beim Prozessor einzeln an den Pins an, sondern sind gemultiplext. Der Pin 24 ist also nicht nur Adreß-Bit 0, sondern auch Adreß-Bit 8. Wie kann das funktionieren?

Die Antwort ist ganz einfach: Der Anschluß ist erst das eine Adreß-Bit, danach das andere. Um jetzt zu einem bestimmten Zeitpunkt sagen zu können, welche Bedeutung der Anschluß hat, werden Hilfssignale benötigt. Diese Hilfssignale heißen -CAS und -RAS. Sie werden unter anderem auch zur Steuerung der dynamischen RAM-Bausteine benötigt, da diese auch einen gemultiplexten Adreßbus aufweisen. Der zeitliche Ablauf des Speicherzugriffs sieht folgendermaßen aus.

Die Signale -CAS und -RAS sind High. Jetzt wird zuerst das niederwertige Adreß-Byte auf den Bus gelegt. Nach kurzer Zeit wird das Signal -RAS Low. Damit wird das Adreß-Byte in die RAMs übernommen und gespeichert. Jetzt ändert sich die Businformation. Aus A0 wird A8, aus A1 wird A9 usw... Wiederum nach kurzer Zeit wird jetzt das Signal -CAS Low. Diese abfallende Flanke wird auf den AM gegeben und erzeugt am Ausgang -CASRAM eine zeitlich geringfügig verzögerte abfallende Flanke. Mit dieser verzögerten Flanke wird nun das High-Byte in die RAMs übernommen. Jetzt liegt die vollständige Adresse vor, und die Daten erscheinen auf dem Datenbus. Diese Vorgänge sind im Timing-Diagramm auf der nächsten Seite noch einmal dargestellt.

Die Schnittstelle zwischen RAM und VIC

Da, wie schon gesagt, der VIC nur die Adreß-Bits A0 bis A13 erzeugt, müssen die für die Adressierung der ganzen 64 K RAM fehlenden Bits zusätzlich erzeugt werden.

Dazu wird der Port A der CIA 2 herangezogen. Die Port-Bits 0 und 1 stellen die Adreß-Bits 14 und 15 dar. Um diese Signale in den Multiplex-Vorgang einzubeziehen, werden sie über das IC U14 geschaltet. Im U14 sind vier invertierende 2:1-Multiplexer integriert. So ein Multiplexer ist in seiner Funktion wohl am einfachsten als Wechselschalter zu sehen. Wahlweise einer von zwei Eingängen wird auf den zugehörigen Ausgang geführt. Im Detail funktioniert die Sache so:

Geschaltet werden die Multiplexer durch das Signal am Eingang S. Liegt an S ein Low, dann sind die Eingänge mit der Bezeichnung A auf den Ausgang durchgeschaltet, liegt S auf High-Pegel, dann sind die B-Eingänge durchgeschaltet.

Die Adreß-Bits A6 und A7 vom VIC liegen am Multiplexer, und zwar A6 an den Eingängen 13 und 14 und A7 an den Eingängen 10 und 11. Wenn jetzt mittels des S-Signals zwischen den Eingängen hin- und hergeschaltet wird, so ist an den Ausgängen keine Änderung festzustellen, da die Adreß-Bits auf beide Eingänge geführt sind. Nur die Polarität der Signale ist an den Ausgängen durch die Inverter-Wirkung der Multiplexer vertauscht. Diese invertierten Adreßsignale werden auf die B-Eingänge der beiden anderen Multiplexer geführt, und zwar -A6 an den Pin 3 und -A7 an den Pin 6. Die A-Eingänge werden mit den genannten Port-Bits der CIA 2 versorgt, Port-Bit 0 als -VA14 an Pin 2, Port-Bit 1 als -VA15 an Pin 5.

Da der S-Eingang von -CAS gesteuert wird, liegt am Ausgang Pin 4 bei -CAS High das nochmals invertierte Adreß-Bit A6, bei -CAS Low das Adreß-Bit A14. Am Ausgang Pin 7 wird entsprechend zwischen -A7 und -VA15 geschaltet. Durch die Invertierung des Multiplexers erscheint dieses Signal als A7 oder A15. Der Pin 15 von U14 ist mit dem Signal AEC verbunden. Er trägt die Bezeichnung -OE, Output Enable. Immer wenn

AEC High ist, werden die Ausgänge des U14 abgeschaltet oder in den sogenannten Tri-State-Zustand versetzt. Dies ist wichtig, da bei AEC High der Prozessor den Bus belegt und seine Adressen über die Multiplexer U13 und U25 auf diesen Bus legt. Nur wenn AEC Low ist, kann der VIC ja den Bus belegen, dann sind die Ausgänge des U14 freigegeben.

16 Farben mit vier Bits, das Farb-RAM

Sollen alle 512 möglichen Zeichen auch noch in 16 verschiedenen Farben dargestellt werden, dann sind vier weitere Daten-Bits erforderlich. Es sind dies die vier Pins 35 bis 38 am VIC. An diesen Pins ist das Color-RAM U6 mit seinen Datenleitungen angeschlossen. Dieses IC ist ein statisches RAM mit 4096 Speicherplätzen. In jedem Speicherplatz kann ein Bit gespeichert werden. Jeweils 4 Speicherplätze werden durch eine Adresse angesprochen. Die Adressierung geschieht zuerst wieder durch das Signal -CS am U6. Wenn dieser Anschluß Low ist, wird das RAM selektiert, die Datenleitungen verlassen den Tri-State-Zustand. Erzeugt wird das -CS-Signal auf zwei verschiedene Arten vom AND-Gatter U27. So seltsam es auch klingen mag, dieses AND-Gatter wird in der Schaltung als ein OR-Gatter betrieben.

Ein AND-Gatter legt den Ausgang dann auf High, wenn alle Eingänge auch High sind. Wenn man jetzt die Logik ein wenig umdreht, kann man auch sagen: Wenn der eine Eingang ODER, der andere Low ist, dann ist der Ausgang auch Low. Diese Betriebsart wird im C64 angewendet.

Das Color-RAM belegt den Adreßbereich von $D800 bis $DBFF. Wenn das Signal AEC High ist, belegt der Prozessor den Bus. Damit ist der eine Eingang des AND-Gatters High. Wenn der Prozessor nicht auf das Color-RAM zugreift, ist der Ausgang -COLOR des Decoders U15 auch High. Damit ist der -CS-Eingang des Color-RAM auf High, das Farb-RAM ist nicht selektiert. Wenn der Prozessor auf das Farb-RAM zugreifen will, legt er die entsprechende Speicheradresse auf den Datenbus. Die Decodierung läuft entsprechend wie die des VIC ab. Nur ist jetzt mit Sicherheit das Adreß-Bit A11 gesetzt. Damit wird der -COLOR-Ausgang des Decoders U15 Low. Jetzt ist ein Eingang

des AND-Gatters Low und entsprechend der Ausgang auch. Damit ist das Farb-RAM selektiert.

Da AEC zu diesem Zeitpunkt High ist, sind die vier Analogschalter im IC U16 geschlossen, die Datenleitungen des Farb-RAM sind mit den vier niederwertigen Datenleitungen des Prozessors verbunden. Damit kann nun das Farb-RAM beschrieben und gelesen werden. Wenn AEC Low wird und der VIC den Bus übernimmt, dann werden die Analogschalter geöffnet. Gleichzeitig wird der Ausgang des AND-Gatters U27 Pin 8 Low, das Farb-RAM ist selektiert, diesmal vom VIC. Da der VIC aber nur mit den Adreßleitungen A8 bis A11 mit dem VIC verbunden ist, müssen die Adreß-Bits A0 bis A7 anders gewonnen werden. Diese Aufgabe übernimmt das IC U26. Dieses TTL-IC mit der Bezeichnung 74LS373 enthält 8 Latches oder Zwischenspeicher. Die Eingänge dieses ICs sind mit dem gemultiplexten Adreßbus verbunden. Eingespeichert werden die Daten, wenn das Signal -RAS Low wird. Das ist der Zeitpunkt, wenn das niederwertige Adreß-Byte auf dem Bus liegt. Die Ausgänge von U16 sind mit dem niederwertigen Adreß-Byte des Prozessorbusses verbunden und liefern die Adreßinformationen, wenn der Prozessor im Tri-State ist.

Auf diese Weise kann der VIC das Farb-RAM adressieren. Auch der Zwischenspeicher ist mit dem Signal AEC verbunden. Am Pin 1 des U16 bewirkt es im High-Zustand, daß die Ausgänge hochohmig werden, um den Prozessor nicht zu stören. Wenn man sich diese Vorgänge genauer anschaut, ergibt sich eine interessante Frage. Wieso hat der VIC zusätzlich zum gemultiplexten Adreßbus A0 bis A13 noch die vier Adreßleitungen A8 bis A11 an den Pins 23, 32, 33 und 34?

Die Antwort ist relativ simpel. Der VIC muß zu jeder Bildschirmspeicheradresse im Bereich von $0400 bis $07FF gleichzeitig die entsprechende Farbspeicherzelle im Adreßbereich $D800 bis $DBFF ansprechen. Dieser gleichzeitige Zugriff auf zwei verschiedene Speicherplätze erfordert einen zweiten, vom normalen Adreßbus unabhängigen Bus. Dieser Bus wird durch die 4 separaten Adreß-Bits realisiert.

Der Prozessor und das RAM

Bisher haben wir uns nur mit dem Fall beschäftigt, daß der VIC auf die 64 K Arbeitsspeicher zugreifen will. Es fehlt noch die Beschreibung der Vorgänge bei einem Zugriff des Prozessors auf dieses RAM. Die Lesezugriffe des Prozessors sind den Zugriffen des VIC sehr ähnlich. In beiden Fällen liegt das Signal R/-W (Lesen bei High, Schreiben bei Low) auf High. Zuerst darum die Lesezugriffe.

Wie bei der Beschreibung der RAM-VIC-Schnittstelle erläutert, benötigt das RAM einen gemultiplexten Adreßbus. Diese Forderung der RAMs kann der Prozessor aber nicht erfüllen. Darum ist ein "Multiplexen" mit zusätzlichen ICs notwendig. Diese Multiplexer sind die ICs U13 und U25, zwei 74LS257.

Diese ICs arbeiten nach demselben Prinzip wie das U14 (beschrieben im Abschnitt RAM und VIC). Der Unterschied zu U14 besteht darin, daß diese Multiplexer die Ausgangssignale nicht invertieren. An den Eingängen der beiden Multiplexer-ICs liegt der komplette Prozessor-Adreßbus A0 bis A15. Dabei sind die Eingänge so geschaltet, daß mit dem Select-Signal jeweils zwischen A0 und A8, A1 und A9 usw. umgeschaltet wird. Die Adressierung der RAMs läßt sich wieder in drei Phasen zerlegen:

In der ersten Phase liegt am Select-Eingang der Multiplexer ein High. Damit ist das niederwertige Adreß-Byte auf die RAMs geschaltet. Mit der abfallenden Flanke des -RAS-Signals wird dieses Byte in die RAMs übernommen. Kurze Zeit später wird auch das -CAS-Signal Low. Damit schalten die Multiplexer um, der jeweils zweite Eingang der Multiplexer wird auf die entsprechenden Ausgänge geschaltet und das höherwertige Adreß-Byte liegt an den RAMs. Über den AM wird das Signal -CAS wieder etwas verzögert. Der Ausgang -CASRAM übernimmt auch hier die eigentliche Funktion des Signals -CAS. Mit der abfallenden Flanke vom -CASRAM wird nun das High-Byte der Adresse in den RAMs gespeichert. Jetzt wird in den RAMs die adressierte Speicherzelle angesprochen, und die Daten erscheinen auf dem Datenbus.

Die Schreibzugriffe des Prozessors unterscheiden sich von den Lesezyklen durch einen wesentlichen Umstand. Bei einem Schreibzugriff wird der Prozessor-Pin R/-W Low, nachdem der Prozessor die Adresse der entsprechenden Speicherzelle auf den Adreßbus gelegt hat. Damit wird dem RAM signalisiert, daß das auf dem Datenbus liegende Byte in dieser Speicherzelle gespeichert werden soll.

Die verwendeten RAM-Bausteine stellen an dieses R/-W-Signal eine bestimmte Bedingung. Das Signal R/-W darf erst dann Low werden, nachdem -RAS Low geworden ist, -CASRAM aber noch High ist. R/-W muß also zwischen den beiden abfallenden Flanken von -RAS und -CASRAM Low werden. Der zeitliche Verlauf der -RAS-, -CAS- und -CASRAM-Signale ist mit denen bei Lesezugriffen identisch.

Der SID 6581, ein Synthesizer mit 28 Beinen und mehr

Dieses IC ist genau wie der VIC ein Paradebeispiel für die Möglichkeiten der Halbleiterindustrie. Durch dieses IC erhält der CBM 64 seine fantastischen Klangmöglichkeiten.

Vor wenigen Jahren hätte allein ein Synthesizer mit diesen in einem IC integrierten Möglichkeiten die ganze Leiterplatte des 64 für sich in Anspruch genommen.

Die 28 Pins des 6581 haben die folgenden Bezeichnungen:

Pin	Bezeichnung	Funktion
1	CAP1A	Externer Kondensator für Frequenzfilter
2	CAP1B	Wie Pin 1
3	CAP2A	Wie Pin 1
4	CAP2B	Wie Pin 1
5	-RES	Eingang, RESET-Signal
6	02	Eingang, Taktsignal
7	R/-W	Eingang, Read/-Write
8	-CS	Eingang, Chip Select
9	A0	Eingang, Adreß-Bit 0
bis		
13	A4	Eingang, Adreß-Bit 4
14	GND	Betriebsspannung Masse

Pin	Bezeichnung	Funktion
15	D0	Daten-Bit 0, bidirektional
bis		
22	D7	Daten-Bit 7, bidirektional
23	POTY	Eingang, AD-Wandler 2
24	POTX	Eingang, AD-Wandler 1
25	Vcc	Betriebsspannung +5V
26	EXT IN	Eingang, externe Signalquelle
27	AUDIO OUT	Ausgang Synthesizer
28	Vdd	Betriebsspannung +12V

Nicht vorgekommen sind bisher die Bezeichnungen der ersten vier Pins, CAP1A bis CAP2B. Wie man im Schaltbild sehen kann, sind an diesen Anschlüssen die 2 Kondensatoren C10 und C11 angeschlossen. Diese Kondensatoren werden für die im Chip U18, dem SID, integrierten Frequenzfilter benötigt.

Ein Filter ist eine uns allen bekannte Einrichtung. Nehmen wir zum Beispiel mal den "Kaffeefilter". Die Aufgabe dieses Filters ist es, bestimmte Anteile (nämlich das Wasser und die löslichen Stoffe des Kaffeepulvers) durchzulassen und andere Anteile (in unserem Beispiel die Reste des Kaffeepulvers) zurückzuhalten.

Genau so arbeitet auch ein elektronischer Frequenzfilter. Bestimmte Frequenzen werden durchgelassen, andere werden zurückgehalten. Es gibt insgesamt vier mögliche Arten von Frequenzfiltern: den Tiefpaß, den Hochpaß, den Bandpaß und den Sperrpaß. Ein Tiefpaß läßt nur tiefe Frequenzen bis zu einer bestimmten höchsten Frequenz passieren. Diese Funktion ist an jeder Stereoanlage in Form des Baßreglers zu finden. Mit diesem Regler läßt sich diese höchste, durchzulassende Frequenz, die sogenannte Grenzfrequenz, einstellen.

Ein Hochpaß zeigt genau das umgekehrte Verhalten, ab einer bestimmten niedrigsten Frequenz läßt er alle höheren Frequenzen durch. Das ist an der Stereoanlage der Trebble- oder Höhenregler.

Bleiben noch Bandpaß und Sperrpaß. Auch diese haben genau entgegengesetzte Funktionen. Ein Bandpaß ist eine Mischung aus Tief- und Hochpaß. Ab einer bestimmten Frequenz werden hö-

here Frequenzen durchgelassen, dies aber nur bis zu einer höchsten Frequenz. Darüberliegende Frequenzen werden wieder gesperrt. Ein Sperrfilter sperrt in einem bestimmten Frequenzbereich alle Frequenzen. Diese Funktion ist an manchen guten Stereoanlagen als Brummfilter vorhanden. Damit wird in diesem Fall nur eine bestimmte Frequenz, die 50 Hertz der Netzfrequenz, herausgefiltert. Alle diese Filter lassen sich im SID programmieren.

Der Pin 5, der RESET-Eingang von U18, wird benötigt, um das IC in einen definierten Zustand zu bringen. Wie bereits beschrieben, liegt an diesem Anschluß nach dem Einschalten für ca. 0.5 Sekunden ein Low-Pegel. Damit werden alle Register im 6581 gelöscht. Ohne diesen RESET würden die Register nach dem Einschalten zufällige Werte haben, die Folge wäre ein genauso zufälliges Signal am Audio-Ausgang; der angeschlossene Fernseher oder Verstärker würde nur "Krach" machen.

Aus der Frequenz des Signals 02 werden alle Tonfrequenzen des SID durch Frequenzteilung erzeugt. Gleichzeitig stellt 02 natürlich wie bei allen anderen Peripherie-Bausteinen den Bezugstakt für die Schreib- und Lesezugriffe des Prozessors dar.

Ob die im SID enthaltenen Register beschrieben oder gelesen werden, hängt in bekannter Weise von der Leitung R/-W ab. Bei einem High werden die Register gelesen, bei einem Low wird in die Register geschrieben. Voraussetzung ist natürlich, das der SID auch korrekt adressiert ist. Der Adreßbereich des 6581 liegt von $D400 bis $D7FF. Dieser Adreßbereich wird wie beim VIC durch den AM und die Decoder im IC U15 decodiert. Sobald der Prozessor eine Adresse in diesem Bereich auf den Bus legt und das Signal -CHAREN High ist, wird der Ausgang Pin 5 des 74LS139 und damit auch der -CS-Eingang des SID Low.

Um nun auch die einzelnen Register im SID zu adressieren, werden die 5 Adreßleitungen A0 bis A4 benötigt.

Sind diese Adreß-Bits alle Low und der SID mit -CS selektiert, kann das Register 0 beschrieben oder gelesen werden. Ist nur das Adreß-Bit A0 High, ist Register 1 selektiert usw.

Auf diese Weise lassen sich alle 29 Register ansprechen. Die Datenleitungen D0 bis D7 an den Pins 15 bis 22 sind mit dem Prozessordatenbus verbunden. Solange -CS High ist, befinden sich die Datenleitungen des SID im Tri-State. Wenn der -CS Low wird, entscheidet R/-W, ob die Datenleitungen als Eingang (beim Schreiben der Register) oder als Ausgang (entsprechend beim Lesen) fungieren. Die Anschlüsse POTX und POTY stellen die Eingänge der "AD-Wandler" dar. Bis jetzt ist die Bezeichnung AD noch nicht erläutert worden. Das wollen wir schnell nachholen.

AD-Wandler ist die Abkürzung für Analog-Digital-Wandler. Ein digitaler Wert kennt bekanntlich nur zwei Zustände, entweder High oder Low, im CBM 64 und vielen anderen Digital- und Computerschaltungen durch eine Spannung von +5V als High und 0 Volt als Low signalisiert. Ein analoges Signal ist da nicht so festgelegt, es kann jeden beliebigen Wert dazwischen, darüber und darunter annehmen. Nun ist es aber oft wünschenswert, einen solchen analogen Wert in einen Computer eingeben zu können, um ihn zu verarbeiten. Diese Möglichkeit der Eingabe analoger Werte ist im CBM 64 eingebaut.

Hauptsächlich genutzt werden die AD-Wandler in Verbindung mit den "Paddles", das sind Drehregler, ähnlich den Reglern an Radiogeräten. Ein solcher Regler enthält einen veränderbaren Widerstand, Potentiometer oder kurz "Poti" genannt. Der Widerstandswert des Potis ändert sich mit dem Drehen. Der minimale Widerstand der in den Paddles eingebauten Potis beträgt ca. 100 Ohm, der Maximalwert ca. 500 KOhm. Dazwischen kann der Widerstand theoretisch jeden beliebigen Wert annehmen.

Der AD-Wandler erzeugt aus diesem Widerstandswert ein digitales Signal. In unserem Fall wird ein 8-Bit-Signal erzeugt. Dieses Byte kann aus einem der SID-Register gelesen werden. Die eigentliche AD-Wandlung geschieht mit dem eingestellten Widerstandswert und den Kondensatoren C48 und C93. Diese Kondensatoren werden für 0.25 Millisekunden über die Potis aufgeladen. Wenn die Spannung an den Kondensatoren größer wird als die im SID erzeugte Vergleichsspannung, wird ein Zähler im SID angehalten. Der Zählerstand ist das Maß für den eingestell-

ten Widerstand: Je größer der Widerstand des Potis ist, desto langsamer wird der Kondensator aufgeladen, und die Spannung am Kondensator erreicht die Höhe der Referenzspannung später. Damit kann der Zähler länger laufen, der Zählerwert wird größer.

Ist der Widerstandswert zu hoch (ca. 200 KOhm), dann erreicht die Spannung am Eingang des AD-Wandlers in der Meßzeit nicht die Referenzspannung. Der Zähler läuft dann bis zu seinem Endwert, im AD-Register steht der Wert 255. Wenn der Widerstand aber zu klein wird (ca. 200 Ohm), ist der Kondensator so schnell aufgeladen, daß der Zähler sofort gestoppt wird. Damit steht im Register ein Wert von 0.

Nach Ablauf der Meßzeit von 0.25 Millisekunden werden die Kondensatoren schlagartig über den entsprechenden AD-Eingang entladen. Jetzt wird der Zähler auf 0 gesetzt, und nach weiteren 0.25 Millisekunden startet dann ein neuer Meßzyklus. Somit benötigt ein vollständiger Zyklus 0.5 Millisekunden, in einer Sekunde werden 2000 mal die aktuellen Widerstandswerte gemessen und stehen zur Verfügung.

Um eine Beschädigung der AD-Eingänge zu vermeiden, sollte der Widerstand nicht kleiner als 100 Ohm werden. Sonst werden die bei der Entladung der Kondensatoren auftretenden Ströme zu groß, und die Entladestufe am Eingang kann zerstört werden. Die zwei Eingänge POTX oder POTY liegen aber nicht direkt an einer der verschiedenen Buchsen des 64. Die beiden Eingänge liegen an den Pins 2, 3, 9 und 10 des IC U28. Dieses IC, ein CMOS-Baustein mit der Bezeichnung 4066, enthält vier sogenannte Analogschalter. Dieses wird benötigt, da an den 64 zwei Paddle-Paare, insgesamt also vier Potis, angeschlossen werden können.

So ein Analogschalter arbeitet vergleichbar einem Relais. Wenn am Steuereingang eine Spannung anliegt, wird der Analogeingang auf den Ausgang durchgeschaltet, der Schalter ist geschlossen. Liegt der Steuereingang auf Masse, dann ist der Ausgang vom Eingang gesperrt, der Schalter ist geöffnet.

Die Analogeingänge sind mit den Controll-Ports CN8 und CN9 verbunden. An diesen Controll-Ports sind die Kontakte 5 und 9 für den Anschluß der Paddles vorgesehen. Die Steuereingänge sind die Pins 5, 6, 12 und 13. Der Pin 13 kontrolliert den Schalter 1 zwischen den Anschlüssen 1 und 2, Pin 5 den Schalter 2 zwischen 4 und 3, Pin 6 den Schalter 3 zwischen 8 und 9 und Pin 12 den Schalter 4 zwischen 11 und 10. Jeweils zwei dieser Eingänge sind zusammengeschaltet, un zwar Pin 13 und 5 und Pin 6 und 12.

Diese jeweils verbundenen Eingänge liegen an den beiden Pins 8 und 9 der CIA U1. Über diese Leitungen kann man auswählen, welche Potis an den Eingängen des AD-Wandlers liegen. Sind die Pins 8 und 9 der CIA U1 Low, dann liegt kein Poti an den Wandlern. Ist Pin 8 High, dann sind die Analogschalter 3 und 4 geschlossen, die am Controll-Port 1 angeschlossenen Paddles werden an die Anschlüsse POTX und POTY gelegt. Ist dagegen Pin 9 der CIA High, dann sind Analogschalter 1 und 2 geschlossen, die Paddles an CN8 liegen an den Eingängen der AD-Wandler. Bleiben noch die Anschlüsse EXT IN und AUDIO OUT am 6581.

AUDIO OUT ist der NF-Ausgang des Synthesizers. Hier stehen die im Synthesizer erzeugten Töne und Geräusche zur Verfügung. Bei maximaler Lautstärke hat das Ausgangssignal eine Größe von 2Vss. Der Transistor Q8 ist als Emitter-Folger an den Ausgang geschaltet. Dadurch, daß das Signal am Emitter des Transistors über dem Widerstand R38 abgenommen wird, hat der Transistor keine Spannungsverstärkung. Das Signal am Ausgang Pin 3 der 8-poligen Video-Audio-Buchse CN5 hat somit auch eine Höhe von 2Vss.

An diesen Ausgang kann man direkt einen kleinen 8-Ohm-Lautsprecher anschließen. Allerdings ist die Lautstärke sehr gering. Um eine vernünftige Wiedergabe zu erreichen, geben Sie das Signal am besten auf eine Stereoanlage oder ein gutes Kofferradio. Oder Sie benutzen den im Fernseher eingebauten Lautsprecher und das mit dem Bild übertragene Tonsignal.

EXT IN gibt die Möglichkeit, auch externe Signale in den Synthesizer einzuspeisen und zu beeinflussen. Externe Signale können beispielsweise Mikrophonsignale sein, die mit einem kleinen Verstärker verstärkt worden sind. Auch eine Gitarre oder eine Orgel kann nach entsprechender Verstärkung das Eingangssignal liefern, aber auch ein zweiter SID, also ein zweiter CBM 64. Damit hätte man dann noch wesentlich mehr Möglichkeiten der Klanggestaltung. Die einzige an das Eingangssignal gestellte Forderung lautet, daß das Signal nicht größer als 3Vss sein darf. Dieser Eingang ist über den Kondensator C12 mit dem Kontakt 5 der 8-poligen Audio-Video-Buchse CN5 verbunden.

Anhang C.11: Floppy-Fehlermeldungen

Die folgenden Fehlermeldungen erhalten Sie durch Auslesen des Floppy-Fehlerkanals. TT steht jeweils für die Tracknummer, SS für die Sektornummer eines Datenblocks.

00,OK,00,00

Kein Fehler aufgetreten.

01,FILES SCRATCHED,XX,00

Rückmeldung nach SCRATCH-Befehl (XX=Anzahl der gelöschten Dateien).

20,READ ERROR,TT,SS

Kopf eines Datenblocks wurde nicht gefunden.

21,READ ERROR,TT,SS

SYNC-Markierung eines Datenblocks wurde nicht gefunden.

22,READ ERROR,TT,SS

Prüfsummenfehler im Kopf eines Datenblocks.

23,READ ERROR,TT,SS

Prüfsummenfehler im Datenteil eines Datenblocks.

24,READ ERROR,TT,SS

Prüfsummenfehler.

25,WRITE ERROR,TT,SS

Datenblock wurde fehlerhaft auf Disk geschrieben.

26,WRITE PROTECT ON,TT,SS

Schreibversuch auf Diskette mit aktiviertem Schreibschutz.

27,READ ERROR,TT,SS

Prüfsummenfehler.

28,WRITE ERROR,TT,SS

SYNC-Markierung eines Datenblocks nicht gefunden.

29,DISK ID MISMATCH,TT,SS

Disk-ID ist fehlerhaft.

30,SYNTAX ERROR,00,00

Gesendeter DOS-Befehl ist fehlerhaft.

31,SYNTAX ERROR,00,00

Gesendeter DOS-Befehl wurde nicht erkannt.

32,SYNTAX ERROR,00,00

Gesendeter DOS-Befehl ist länger als 40 Zeichen.

33,SYNTAX ERROR,00,00

Unzulässige Verwendung des Jokers.

34,SYNTAX ERROR,00,00

In einem gesendeten DOS-Befehl wurde der File-Name nicht gefunden.

39,FILE NOT FOUND,00,00

Programm vom Typ "USR" wurde nicht gefunden.

50,RECORD NOT PRESENT,00,00

Es wurde ein Datensatz einer relativen Datei angesprochen, der noch nicht existiert. Sofern ein Schreibzugriff erfolgt ist, wurde der Datensatz angelegt.

51,OVERFLOW IN RECORD,00,00

Datensatzlänge einer relativen Datei ist zu klein für die gesendeten Daten.

52,FILE TOO LARGE,00,00

Datensatznummer einer relativen Datei ist zu groß.

60,WRITE FILE OPEN,00,00

Es wurde versucht, eine Datei zu öffnen, die beim letzten Schreibvorgang nicht ordnungsgemäß geschlossen wurde.

61,FILE NOT OPEN,00,00

Eine angesprochene Datei wurde zuvor nicht geöffnet.

62,FILE NOT FOUND,00,00

Die angesprochene Datei existiert auf der Diskette nicht.

63,FILE EXISTS,00,00

Es wurde versucht, eine Datei mit einem auf der Diskette bereits existierenden Namen anzulegen.

64,FILE TYPE MISMATCH,00,00

Beim Öffnen einer Datei wurde ein falscher Dateityp angegeben.

65,NO BLOCK,TT,SS

Es wurde versucht, einen Datenblock zu belegen, der bereits nicht mehr frei war.

66,ILLEGAL TRACK OR SECTOR,TT,SS

Es wurde versucht, auf einen nicht existierenden Datenblock zuzugreifen.

67,ILLEGAL TRACK OR SECTOR,TT,SS

Die Track-Sektor-Verkettung einer Datei ist fehlerhaft.

70,NO CHANNEL,00,00

Es wurde versucht, mehr Dateien zu öffnen als Kanäle vorhanden sind.

71,DIR ERROR,TT,SS

Die Anzahl der freien Datenblöcke im DOS-Speicher stimmt mit dem Bit-Muster der BAM nicht überein.

72,DISK FULL,00,00

Diskette ist komplett mit Daten belegt.

73,CBM DOS V2.6 1541,00,00

Einschaltmeldung der Floppy.

74,DRIVE NOT READY,00,00

Es wurde versucht, die Floppy anzusprechen, ohne daß sich eine Diskette im Laufwerk befindet.

Anhang C.12: Bildschirmcodes

Die untenstehende Tabelle enthält eine vollständige Liste der POKE/PEEK-Codes und der CHR$/ASC-Codes.



Dez	Hex	POKE- Zeichen Gross	POKE- Zeichen Klein	CHR$- Zeichen Gross	CHR$- Zeichen Klein
000	$00	@	@	Keine Funktion	
001	$01	A	a	Keine Funktion	
002	$02	B	b	Keine Funktion	
003	$03	C	c	Keine Funktion	
004	$04	D	d	Keine Funktion	
005	$05	E	e	Weiss	
006	$06	F	f	Keine Funktion	
007	$07	G	g	Keine Funktion	
008	$08	H	h	Shift Commodore verboten	
009	$09	I	i	Shift Commodore erlauben	
010	$0a	J	j	Keine Funktion	
011	$0b	K	k	Keine Funktion	
012	$0c	L	l	Keine Funktion	
013	$0d	M	m	Return	
014	$0e	N	n	Kleinbuchstaben	
015	$0f	O	o	Keine Funktion	
016	$10	P	p	Keine Funktion	
017	$11	Q	q	Cursor abwaerts	
018	$12	R	r	Revers anschalten	
019	$13	S	s	Cursor Home	
020	$14	T	t	Delete	
021	$15	U	u	Keine Funktion	
022	$16	V	v	Keine Funktion	
023	$17	W	w	Keine Funktion	
024	$18	X	x	Keine Funktion	
025	$19	Y	y	Keine Funktion	
026	$1a	Z	z	Keine Funktion	
027	$1b	[	[	Escape	
028	$1c	£	£	Rot	
029	$1d	]	]	Cursor rechts	
030	$1e	↑	↑	Gruen	
031	$1f	←	←	Dunkelblau	
032	$20			Space	
033	$21	!	!	!	!
034	$22			"	"
035	$23	#	#	#	#
036	$24	$	$	$	$
037	$25	%	%	%	%
038	$26	&	&	&	&
039	$27	'	'	'	'
040	$28	(	(	(	(
041	$29	)	)	)	)
042	$2a	*	*	*	*
043	$2b	+	+	+	+
044	$2c	,	,	,	,
045	$2d	—	—	—	—
046	$2e	.	.	.	.
047	$2f	/	/	/	/
048	$30	0	0	0	0
049	$31	1	1	1	1
050	$32	2	2	2	2
051	$33	3	3	3	3
052	$34	4	4	4	4
053	$35	5	5	5	5
054	$36	6	6	6	6
055	$37	7	7	7	7
056	$38	8	8	8	8
057	$39	9	9	9	9
058	$3a	:	:	:	:
059	$3b	;	;	;	;

060	$3c	<	<	<	<
061	$3d	=	=	=	=
062	$3e	>	>	>	>
063	$3f	?	?	?	?
064	$40			@	@
065	$41		A	A	a
066	$42		B	B	b
067	$43		C	C	c
068	$44		D	D	d
069	$45		E	E	e
070	$46		F	F	f
071	$47		G	G	g
072	$48		H	H	h
073	$49		I	I	i
074	$4a		J	J	j
075	$4b		K	K	k
076	$4c		L	L	l
077	$4d		M	M	m
078	$4e		N	N	n
079	$4f		O	O	o
080	$50		P	P	p
081	$51		Q	Q	q
082	$52		R	R	r
083	$53		S	S	s
084	$54		T	T	t
085	$55		U	U	u
086	$56		V	V	v
087	$57		W	W	w
088	$58		X	X	x
089	$59		Y	Y	y
090	$5a		Z	Z	z
091	$5b			[	[
092	$5c			£	£
093	$5d			]	]
094	$5e			↑	↑
095	$5f			←	←
096	$60				
097	$61				A
098	$62				B
099	$63				C
100	$64				D
101	$65				E
102	$66				F
103	$67				G
104	$68				H
105	$69				I
106	$6a				J
107	$6b				K
108	$6c				L
109	$6d				M
110	$6e				N
111	$6f				O
112	$70				P
113	$71				Q
114	$72				R
115	$73				S
116	$74				T
117	$75				U
118	$76				V
119	$77				W
120	$78				X
121	$79				Y
122	$7a				Z
123	$7b				
124	$7c				
125	$7d				
126	$7e				
127	$7f				
128	$80				Keine Funktion
129	$81				Keine Funktion
130	$82				Keine Funktion
131	$83				Keine Funktion

132	$84			Keine Funktion	
133	$85			Weiss	
134	$86			Keine Funktion	
135	$87			Keine Funktion	
136	$88			Shift Commodore verboten	
137	$89			Shift Commodore erlauben	
138	$8a			Keine Funktion	
139	$8b			Keine Funktion	
140	$8c			Keine Funktion	
141	$8d			Return	
142	$8e			Kleinbuchstaben	
143	$8f			Keine Funktion	
144	$90			Keine Funktion	
145	$91			Cursor abwaerts	
146	$92			Revers anschalten	
147	$93			Cursor Home	
148	$94			Delete	
149	$95			Keine Funktion	
150	$96			Keine Funktion	
151	$97			Keine Funktion	
152	$98			Keine Funktion	
153	$99			Keine Funktion	
154	$9a			Keine Funktion	
155	$9b			Escape	
156	$9c			Rot	
157	$9d			Cursor rechts	
158	$9e			Gruen	
159	$9f			Dunkelblau	
160	$a0			Space	
161	$a1			!	!
162	$a2			"	"
163	$a3			#	#
164	$a4			$	$
165	$a5			%	%
166	$a6			&	&
167	$a7			'	'
168	$a8			(	(
169	$a9			)	)
170	$aa			*	*
171	$ab			+	+
172	$ac			,	,
173	$ad			—	—
174	$ae			.	.
175	$af			/	/
176	$b0			0	0
177	$b1			1	1
178	$b2			2	2
179	$b3			3	3
180	$b4			4	4
181	$b5			5	5
182	$b6			6	6
183	$b7			7	7
184	$b8			8	8
185	$b9			9	9
186	$ba			:	:
187	$bb			;	;
188	$bc			<	<
189	$bd			=	=
190	$be			>	>
191	$bf			?	?
192	$c0			@	@
193	$c1			A	a
194	$c2			B	b
195	$c3			C	c
196	$c4			D	d
197	$c5			E	e
198	$c6			F	f
199	$c7			G	g
200	$c8			H	h
201	$c9			I	i
202	$ca			J	j
203	$cb			K	k

204	$cc			L	l
205	$cd			M	m
206	$ce			N	n
207	$cf			O	o
208	$d0			P	p
209	$d1			Q	q
210	$d2			R	r
211	$d3			S	s
212	$d4			T	t
213	$d5			U	u
214	$d6			V	v
215	$d7			W	w
216	$d8			X	x
217	$d9			Y	y
218	$da			Z	z
219	$db			[	[
220	$dc			£	£
221	$dd			]	]
222	$de			↑	↑
223	$df			←	←
224	$e0				
225	$e1				A
226	$e2				B
227	$e3				C
228	$e4				D
229	$e5				E
230	$e6				F
231	$e7				G
232	$e8				H
233	$e9				I
234	$ea				J
235	$eb				K
236	$ec				L
237	$ed				M
238	$ee				N
239	$ef				O
240	$f0				P
241	$f1				Q
242	$f2				R
243	$f3				S
244	$f4				T
245	$f5				U
246	$f6				V
247	$f7				W
248	$f8				X
249	$f9				Y
250	$fa				Z
251	$fb				
252	$fc				
253	$fd				
254	$fe				
255	$ff				

Anhang D: Die abgedruckten Programme

Die folgende Aufstellung enthält eine Übersicht über die in den einzelnen Kapiteln abgedruckten BASIC- und Assembler-Programme. Dabei habe ich mich auf jene Programme beschränkt, die einen unmittelbaren praktischen Nutzen haben, also über reine Beispiele hinausgehen.

Anhang D.1: BASIC-Programme

Kapitel 5

ADRESSVERWALTUNG

Verwaltet Adressen auf indexsequentieller Basis.

Kapitel 6

KREIS

Zum Zeichnen von Kreisen in hochauflösender Grafik.

ELLIPSE

Zum Zeichnen von Ellipsen in hochauflösender Grafik.

RECHTECK

Zeichnet Rechtecke in hochauflösender Grafik.

POLYGON

Zeichnet Polygonzüge in hochauflösender Grafik.

SPRITE-EDIT

Editor zum Entwerfen von Sprites.

CHAR-EDIT

Editor zum Entwerfen eigener Zeichen.

Kapitel 7

MUSIK

Zum Abspielen von Musikstücken.

Kapitel 9

SPUR1

Zum Erstellen einer Spureinstellungs-Diskette.

SPUR2

Zur Spureinstellung.

SCHREIB-LESE-TEST

Testet einzelne Spuren auf defekte Sektoren.

GESCHWINDIGKEIT

Überprüft die Umdrehungsgeschwindigkeit einer Diskette.

Kapitel 10

INHALTSVERZEICHNIS

Gibt das Inhaltsverzeichnis einer Kassette aus.

KATALOG

Legt einen Katalog der auf einer Kassette gespeicherten Dateien an.

KOPFJUSTAGE

Hilfsprogramm zur Justierung des Schreib-/Lesekopfes der Datasette.

FT-INHALTSVERZEICHNIS

Gibt das Inhaltsverzeichnis von FastTape-Kassetten aus.

Anhang D.2: Assembler-Programme

Hinweis: Die im folgenden erwähnten Assembler-Programme liegen alle auch als sog. BASIC-Lader vor (in der Regel jeweils im Anschluß an das Assembler-Listing) und sind daher auch für Nur-BASIC-Programmierer nutzbar!

Kapitel 4

DSAVEM

Speichert einen beliebigen Speicherbereich auf Disk ab.

DLOADM

Lädt einen beliebigen Speicherbereich von Disk in den Rechner.

TRANSFER

Verschiebt beliebige Speicherbereiche.

MYFILL

Füllt beliebige Speicherbereiche mit einem Wert.

FUNKTIONSTASTEN

Routinen zur Belegung der Funktionstasten mit Texten.

Kapitel 5

DGETV

Liest Datenblöcke bis zu einer Länge von 255 Byte von Diskette ein.

Kapitel 6

HIRESGRAFIK

Routinensammlung zur hochauflösenden Grafik.

SPRITES

Routinensammlung zur Sprite-Programmierung.

ZEICHENSATZ

Routinensammlung zur Zeichensatzprogrammierung.

Kapitel 7

SOUND

Umfangreiche Routinensammlung zur Sound-Programmierung.

Kapitel 10

UNNEW

Holt ein mit NEW gelöschtes BASIC-Programm zurück.

SAVE-ADRESSE

Speichert ein Maschinenprogramm auf Kassette.

LADER

Erzeugt ein Lader-Programm zu einem BASIC-Programm.

MERGE

Hängt ein BASIC-Programm hinter ein anderes.

FASTTAPE

Neues Kassetten-Betriebssystem.

DATENVERARBEITUNG

Datenverwaltungsprogramm, das mit FastTape zusammenarbeitet.

BACKUP CC-DISK

Erzeugt ein Backup von Kassette auf Disk.

BACKUP DISK-CC

Erzeugt ein Backup von Disk auf Kassette.

Anhang E: Anbieterverzeichnis

Die folgende Aufstellung enthält ein Verzeichnis der Anbieter, der in Kapitel 2 angesprochenen Hard- und Software-Produkte.

Ein herzlicher Dank geht an die Firmen Scanntronik und Heureka Teachware für die freundliche Bereitstellung ihrer Produkte sowie an das Data Becker-Lektorat für die Bereitstellung eines GEOS-2.0-Exemplars.

Textomat Plus/Datamat/Superbase

Data Becker GmbH
Merowingerstr. 30
4000 Düsseldorf 1

GEOS 2.0 und GEOS-Produkte

Markt und Technik AG
Hans-Pinsel-Str. 2
8013 Haar

Vizawrite

DTM
Bornhofenweg 5
6200 Wiesbaden

Printfox, Pagefox, Eddifox

Scanntronik
Parkstr. 38
8011 Zorneding

Lernsoftware

Heureka Teachware

Ostermann Verlag
Paul-Hösch-Str. 4
8000 München 60

Anhang F: Fachwortlexikon

Das Fachwortlexikon ist zum schnellen Nachschlagen einzelner Fachbegriffe rund um den Commodore 64 gedacht. Die wichtigsten Fachwörter werden kurz erklärt und gegebenenfalls auf die vorangegangenen Kapitel verwiesen.

Sollten Sie einen bestimmten Begriff wider Erwarten einmal nicht vorfinden, so hilft sicher ein Blick in das Stichwortverzeichnis. Der Begriff wurde dann im vorderen Teil des Buches so ausführlich besprochen, daß ich ihn nicht noch einmal extra hier aufgenommen habe.

Anklicken

Unter GEOS haben Sie die Möglichkeit sog. → Icons oder → Menüs (genauer: Pulldown-Menüs) mit der Maus auszuwählen. Dazu müssen Sie den Mauszeiger auf das betreffende Objekt setzen und dann kurz die linke Maustaste drücken. Diesen Vorgang bezeichnet man als "Anklicken" des Objekts.

Zusätzlich gibt es noch einen sog. Doppelklick (damit meint man das zweimalige Drücken der linken Maustaste kurz hintereinander), der zum Beispiel zum Starten von Programmen benötigt wird.

ASCII-Code

ASCII steht für "American Standard Code for Information Interchange", was soviel heißt wie "Amerikanischer Standardcode für Informationsaustausch". Da Computer intern nun einmal nur Zahlen verarbeiten können, müssen alle Zeichen (wie zum Beispiel die Buchstaben des Alphabets) eine bestimmte Codenummer erhalten. (Der Buchstabe "A" beispielsweise hat den Code "65".) Eine vollständige Tabelle des C64-ASCII-Code finden Sie in Anhang.

Assembler

Im Grunde genommen versteht der Commodore 64 nur eine Sprache, die sog. Maschinensprache. Diese besteht nur aus Folgen von Nullen und Einsen. Da damit natürlich selbst Profis nicht vernünftig arbeiten können, hat man den sog. Assembler eingeführt, in dem sich mit symbolischen Namen arbeiten läßt. Befehle an den → Prozessor beispielsweise werden als dreistellige Namen geschrieben. LDA zum Beispiel bedeutet "Lade Akkumulator".

Mit Hilfe von Assembler läßt sich der → Prozessor des Commodore 64 direkt programmieren. BASIC-Programme müssen jeweils zunächst in Maschinensprache übersetzt werden, was in der Regel sehr viel Zeit beansprucht. Bei zeitkritischen Programmen sollte man daher immer auf Assembler zurückgreifen.

BAM

BAM steht für "Block Availability Map" (deutsch: Blockbelegungstabelle). In dieser auf jeder Diskette vorhandenen Tabelle werden die schon belegten und noch freien Datenblöcke der Diskette vermerkt.

Betriebssystem

Das Betriebssystem (auch "Kernal" genannt) stellt die unterste Ebene der → Software eines Rechners dar. Es ist für alle elementaren Vorgänge innerhalb des Rechners (z.B. Tastaturabfrage und Bildschirmausgabe) zuständig, sorgt also für einen ordnungsgemäßen "Betrieb" des Rechners.

Binärsystem

Auch "Dualsystem" genannt (→ Zahlensysteme).

Bit

Ein Bit ist die kleinste Informationseinheit in einem Rechner. Ein Bit kann nur die Werte 0 und 1 annehmen.

Byte

Ein Byte besteht aus acht → Bits. In einem Byte lassen sich Werte von 0 bis 255 darstellen (siehe auch → Zahlensysteme). Jede Speicherzelle im C64 kann genau ein Byte aufnehmen.

Compiler

Ein Compiler übersetzt ein Programm vor der ersten Ausführung komplett in die für den Rechner unmittelbar "verständliche" Maschinensprache. Im Gegensatz zu einem → Interpreter, der ein Programm bei jedem Programmlauf erneut interpretiert, muß das Programm bei einem Compiler also nur einmal analysiert werden, was die Ausführungsgeschwindigkeit des Programms in der Regel beträchtlich erhöht.

Cursor

Der Cursor markiert ganz allgemein die Bildschirmposition, an der Sie sich gerade befinden und zum Beispiel Eingaben tätigen können. Normalerweise wird der Cursor durch ein blinkendes Rechteck dargestellt und kann mit den Cursor-Steuertasten bewegt werden. Unter GEOS gibt es aber auch noch einen sog. Maus-Cursor (auch Mauszeiger genannt), der durch einen schräggestellten Pfeil dargestellt und mit einer Maus bewegt wird.

Datei

Unter einer Datei (engl.: File) versteht man ganz allgemein eine Ansammlung logisch zusammenhängender Daten. Im engeren Sinne meint man damit meistens auf Diskette oder Kassette unter einem bestimmten Namen gespeicherte Daten, wie zum Beispiel Adressen oder Texte, aber auch Programme.

Directory

Unter einem Directory versteht man das Inhaltsverzeichnis einer Diskette. Es enthält Informationen über alle auf der Diskette gespeicherten Dateien. Dazu zählen insbesondere die Namen der Dateien und ihre Länge.

Direktmodus

Im Direktmodus des Commodore 64 werden BASIC-Befehle unmittelbar nach ihrer Eingabe (und dem Drücken der <Return>-Taste) ausgeführt. So lange der Rechner kein Programm abarbeitet, befindet er sich grundsätzlich im Direktmodus, was unter anderem am blinkenden Cursor erkennbar ist.

Editor

Ein Editor ist nichts anderes als eine kleine Textverarbeitung, die auf die Eingabe und Änderung von Programmen spezialisiert ist. Der BASIC-2.0-Editor zum Beispiel erlaubt nicht nur die Eingabe von Programmzeilen, sondern ordnet diese - entsprechend ihrer Zeilennummer - auch gleich richtig in das vorhandene Programm ein.

Expansion-Port

Der Expansion-Port ist eine → Schnittstelle auf der Rückseite des Commodore 64, die zum Anschluß sog. Steckmodule dient.

File

→ Datei

Formatieren

Bevor Sie eine Diskette das erste Mal benutzen, muß diese für die Datenaufnahme speziell vorbereitet werden. Diesen Vorgang bezeichnet man als Formatieren der Diskette. Falls auf der Diskette bereits Daten vorhanden waren, so gehen diese durch das Formatieren unwiderbringlich verloren. Also Vorsicht!

Geräteadresse

Da der Commodore 64 mit Zahlen nun einmal besser umgehen kann als mit Texten, spricht er die an ihn anschließbaren Geräte (auch → Peripherie genannt) durch eine Geräteadresse an. Die Floppy zum Beispiel hat die Geräteadresse 8, der Drucker die Adresse 4.

Hardware

Als Hardware bezeichnet man - im Gegensatz zur → Software - alle "festen" Bestandteile eines Computers. Dazu zählen zum Beispiel der → Prozessor, die einzelnen Chips, die Tastatur oder die Floppy.

Hexadezimalsystem

→ Zahlensysteme

Hochauflösende Grafik

Im normalen Textmodus des Commodore 64 lassen sich auf dem Bildschirm maximal 40x25 Zeichen unterbringen, was eine "Auflösung" von 1.000 Stellen oder Punkten ergibt. Bei der hochauflösenden Grafik dagegen können 320x200 (=64.000!) einzelne Punkte angesprochen werden. Daher kommt die Bezeichnung "hochauflösend".

Icon

Unter einem Icon (zu deutsch: Piktogramm) versteht man ein in der Regel grafisches Symbol, das stellvertretend für etwas anderes, beispielsweise ein Programm, steht. Icons werden auf dem Commodore 64 vor allem von GEOS verwendet und dienen der vereinfachten Bedienung eines Programms.

Interpreter

Ein Interpreter, wie zum Beispiel der BASIC-2.0-Interpreter des Commodore 64, übersetzt ein Programm, im Gegensatz zu einem → Compiler, während der Abarbeitung jedesmal neu. Das hat den Vorteil, daß sich das Programm nach evtl. erforderlichen Änderungen leichter austesten läßt, da es nicht jedesmal zuerst wieder compiliert werden muß. Der vielleicht größte Nachteil eines Interpreters ist die relativ geringe Ausführungsgeschwindigkeit eines Programms, bedingt durch das ständige Übersetzen während der Programmausführung.

Interrupt

Unter einem Interrupt versteht man die Unterbrechung der laufenden Programmabarbeitung durch den → Prozessor. Dazu wird an den Prozessor ein Signal gesendet, das ihn veranlaßt in eine spezielle Interrupt-Routine zu verzweigen. Diese fragt unter anderem die Tastatur ab und sorgt für das Blinken des Cursors. Nach Abarbeitung der Interrupt-Routine fährt der Prozessor mit dem unterbrochenen Programm fort, als ob keine Unterbrechung stattgefunden hätte. Im Commodore 64 wird etwa alle 1/60 Sekunde ein Interrupt ausgelöst.

Jokerzeichen

Jokerzeichen sind ganz allgemein Zeichen, die als Ersatz für beliebige andere Zeichen stehen können. Die Floppy 1541 erlaubt die Verwendung von zwei Jokerzeichen (# und ?). Damit lassen sich zum Beispiel mehrere Dateien gleichzeitig löschen. "S:TE???" etwa würde alle Dateien, die mit "TE" beginnen und fünf Zeichen lang sind, löschen.

Kernel

→ Betriebssystem

Menü

Anwendungsprogramme bieten ihre Funktionen meist in Form sog. Menüs dar. Ähnlich wie auf einer Speisekarte können Sie aus dieser Auswahlliste ablesen, welche Funktionen Ihnen momentan zur Verfügung stehen, und die gewünschte Funktion dann zum Beispiel durch Eingabe eines Kennbuchstabens aufrufen.

Eine spezielle Form von Menüs bietet GEOS an: die sog. Pulldown-Menüs. Durch Anklicken eines Hauptmenü-Punktes mit der Maus wird ein Untermenü "heruntergerollt" (daher der Name "Pulldown"), aus dem - wiederum durch Anklicken mit der Maus - die gewünschte Funktion angewählt wird.

Monitor

Wenn man von einem Monitor spricht, meint man in der Regel das Sichtgerät zur Darstellung der vom Computer gesendeten Ausgaben. Beim Commodore 64 kann es sich dabei auch um einen Fernseher handeln. Der Begriff "Monitor" hat im Computerbereich aber auch noch eine andere Bedeutung: Ein Monitor ist ein Programm, mit dem man in den Speicher eines Rechners "schauen", sich also die Inhalte der einzelnen Speicherzellen zeigen lassen kann. Gute Monitorprogramme bieten zudem die Möglichkeit, Assembler-Programme auszutesten. Ein Monitor ist daher die ideale Ergänzung zu einem → Assembler.

Oktalsystem

→ Zahlensysteme

Parallel

Parallel bezeichnet eine bestimmte Art der Datenübertragung, bei der mehrere Bits (meist acht) gleichzeitig übertragen werden. Im Vergleich zur → seriellen Übertragung ist die parallele Datenübertragung um einiges schneller.

Parameter

Fast jede Anweisung einer → Programmiersprache benötigt eine bestimmte Anzahl von Daten, um korrekt arbeiten zu können. Diese Daten bezeichnet man als Parameter der Anweisung. Der INPUT-Befehl des BASIC-2.0 beispielsweise benötigt den Namen der Variablen, an die die Eingabe übergeben werden soll.

Peripherie

Mit Peripherie bezeichnet man sämtliche an einen Computer anschließbaren Zusatzgeräte. Dazu zählen insbesondere der Monitor, die Datasette, die Floppy sowie der Drucker.

Piktogramm

→ Icon

Pixel

Ein Pixel ist der kleinste darstellbare Punkt auf dem Bildschirm. Die Größe eines Pixels hängt von der eingestellten Grafikauflösung ab. Im Textmodus des Commodore 64 können nur 1.000 Pixel dargestellt werden. Die einzelnen Pixel sind daher entsprechend groß. Man spricht hier auch von einer sog. Blockgrafik. Bei der → hochauflösenden Grafik dagegen lassen sich 64.000 Pixel darstellen, wobei dann ein einzelnes Pixel sehr klein ist.

Programmiersprache

Im wesentlichen gibt es drei Arten von → Software. Da ist zunächst einmal das → Betriebssystem des Computers, das alle elementaren Abläufe innerhalb des Rechners steuert. Mit Anwendungsprogrammen lassen sich spezielle Aufgaben erledigen, mit einer Textverarbeitung etwa das Schreiben von Briefen. Die dritte Sorte Software stellen die sog. Programmiersprachen dar. Mit ihnen kann man selbst Anwendungsprogramme schreiben. Auf dem Commodore 64 stehen die Programmiersprachen BASIC und Assembler zur Verfügung.

Prozessor

Der Prozessor 6510 bildet das Herzstück des Commodore 64. Er steuert sämtliche Abläufe innerhalb des Computers durch die Abarbeitung sog. Maschinenprogramme. Mit Hilfe eines → Assemblers läßt sich der Prozessor vom Anwender direkt programmieren. BASIC-Programme müssen jeweils zunächst in Maschinenprogramme übersetzt werden, was viel Zeit kostet.

Puffer

Unter einem Puffer versteht man in der Regel einen Zwischenspeicher für Daten. Oft ist es aus zeitlichen Gründen günstiger, Daten so lange in einem Puffer abzulegen, bis sich eine größere Datenmenge angesammelt hat, um sie dann anschließend zusammen weiterzuverarbeiten.

RAM

RAM steht für "Random Access Memory". Gemeint ist damit derjenige Speicher im Commodore 64, der frei beschrieben und gelesen werden kann. Der Commodore 64 verfügt über 64 KBytes an RAM-Speicher, der aber nicht vollständig genutzt werden kann.

Register

Sowohl der → Prozessor als auch die verschiedenen Spezialbausteine des Commodore 64 verfügen über sog. Register, die entweder zur Aufnahme und Verarbeitung von Daten oder zur Steuerung bestimmter Vorgänge dienen. So lassen sich zum Beispiel über ein bestimmtes Register des Videochips die sog. → Sprites ein- und ausschalten.

ROM

ROM steht für "Read Only Memory". Gemeint ist damit derjenige Speicher im Commodore 64, der nur gelesen werden kann. Im ROM-Speicher stehen das → Betriebssystem und der BASIC-2.0-Interpreter.

Routine

Unter einer Routine versteht man in der Regel einen kleinen, in sich abgeschlossenen Programmteil, der eine häufig benötigte Programmfunktion ausführt und deshalb von mehreren Stellen des Hauptprogramms (oder von einem anderen Programm aus) aufgerufen wird.

Schnittstelle

Eine Schnittstelle bildet das Verbindungsstück zwischen dem Computer und einem Peripheriegerät, beispielsweise der Floppy oder dem Drucker. Im einfachsten Fall besteht diese Schnittstelle nur aus einem Kabel. Oft ist der Aufwand aber um einiges größer, da es zwischen Peripheriegerät und Computer mitunter zu erheblichen "Verständigungsschwierigkeiten" kommt, die dann

durch die Schnittstelle ausgebügelt werden müssen. Am häufigsten tritt dieses Problem beim Anschluß eines Druckers auf.

Sekundäradresse

Die Sekundäradresse ist im Zusammenhang mit der → Geräteadresse von Bedeutung. Sie dient zur Übermittlung zusätzlicher Informationen bei der Kommunikation mit Peripheriegeräten. So lassen sich beispielsweise die meisten Drucker durch Angabe der entsprechenden Sekundäradresse auf eine bestimmte Schriftart umschalten.

Seriell

Seriell bezeichnet eine bestimmte Art der Datenübertragung, bei der die einzelnen Bits nacheinander übertragen werden. Im Vergleich zur → parallelen Übertragung ist die serielle Datenübertragung relativ langsam, dafür aber sicherer.

Software

Software bezeichnet - im Gegensatz zur → Hardware - all jene Bestandteile eines Computers, die der Hardware "sagen", was sie tun soll. Beispiele für Software sind das → Betriebssystem, → Programmiersprachen sowie Anwendungsprogramme, wie etwa Textverarbeitungen oder Grafikprogramme.

Sprites

Sprites sind eine spezielle Art von Grafikobjekten, die unabhängig von der sonstigen Bildschirmdarstellung frei auf dem Bildschirm bewegt werden können.

Stack

Der Stack (deutsch: Stapel) ist ein spezieller Speicherbereich im Commodore 64, der vor allem für das Ablegen von Rücksprung-Adressen aller Art verwendet wird. Daten, die auf den Stack gelegt werden, werden wie auf einem "natürlichen" Stapel (ein Stapel Papier etwa) behandelt, d.h., das Datum, das zuletzt auf

den Stapel gelegt wurde, wird dann natürlich auch als erstes wieder vom Stapel heruntergeholt.

Steuerzeichen

Als Steuerzeichen bezeichnet man spezielle Zeichen, die im Gegensatz zu normalen Zeichen, wie etwa Buchstaben, bei ihrer Ausgabe einen bestimmten Vorgang auslösen, beispielsweise den Bildschirm löschen oder den Drucker auf eine andere Schriftart umschalten. Siehe dazu auch die ASCII-Tabelle in Anhang.

String

Bei einem String handelt es sich um eine Zeichenkette, die aus beliebigen Zeichen bestehen darf. Im Gegensatz zu Zahlendaten haben Strings im allgemeinen eine variable Länge.

Syntax

Als Syntax einer → Programmiersprache bezeichnet man die Regeln, die angeben, wie die einzelnen Elemente der Sprache angeordnet werden dürfen. Eine Syntaxregel des BASIC-2.0 besagt zum Beispiel, daß zwischen zwei Befehlen ein Doppelpunkt stehen muß.

Taktzyklus

Der Taktzyklus ist die zentrale Zeiteinheit für den → Prozessor des Commodore 64. Innerhalb eines Taktzyklus kann der Prozessor genau eine bestimmte interne Operation durchführen. Die Ausführungszeit von → Assembler-Befehlen wird normalerweise in Taktzyklen angegeben. Ein Taktzyklus auf dem Commodore 64 dauert etwa 1/1.000.000 Sekunde.

User-Port

Der User-Port ist eine → Schnittstelle auf der Rückseite des Commodore 64, die zum Anschluß spezieller Hardware-Erweiterungen dient.

Zahlensysteme

Im Computerbereich gibt es drei wichtige Zahlensysteme: Das Dezimalsystem, das Binärsystem sowie das Hexadezimalsystem. Das Dezimalsystem mit seiner Basis 10 dürfte Ihnen geläufig sein. Das Binärsystem hat die Basis 2 und erlaubt nur die Ziffern 0 und 1. Eine dezimale 15 beispielsweise wird im Binärsystem als 1111 geschrieben. Das Hexadezimalsystem besitzt die Basis 16. Zur Darstellung der Ziffern "10" bis "15" werden die Buchstaben "A" bis "F" verwendet. Eine dezimale 30 wird im Hexadezimalsystem daher als 1E geschrieben.

Zeichensatz

Als Zeichensatz bezeichnet man die Gesamtheit aller auf dem Commodore 64 darstellbaren Zeichen. Der Zeichensatz läßt sich vom Anwender bei Bedarf auch abändern, etwa um die deutschen Umlaute einzubauen!

Zeropage

Der gesamte Speicher des Commodore 64 ist in sog. "Pages" (deutsch: Seiten) eingeteilt. Eine Page umfaßt jeweils genau 256 Byte, was unter anderem mit der Architektur des → Prozessors zusammenhängt. Eine besondere Stellung nimmt die Page Null, die "Zeropage", ein. Da der Prozessor auf sie besonders leicht und schnell zugreifen kann, sind in ihr zahlreiche wichtige Daten gespeichert, beispielsweise die Zeiger auf den BASIC-Speicher.

Anhang G: Quellennachweis:

Kapitel 1.8 -1.8.1, Kapitel 2.7.1, 2.7.2, Kapitel 10.14 - 10.14.5, Anhang C.11, C.12
Polk
Die besten Tips & Tricks, 2. Auflage 1989, DATA BECKER

Kapitel 1.8 - 1.8.8
Tornsdorf
GEOS für Einsteiger, 1. Auflage 1988, DATA BECKER

Kapitel 3.7
Liesert
Peeks & Pokes zum Commodore 64, 3. Auflage 1986, DATA BECKER

Kapitel 3.7
Tornsdorf & Tornsdorf
C64 BASIC für Einsteiger, 3. Auflage 1989, DATA BECKER

Kapitel 4.11, Kapitel 9.4, Anhang C.6 bis C.10
Brückmann, Englisch, Felt, Gelfand, Gerits, Krsnik
64 Intern, 7. Auflage 1988, DATA BECKER

Kapitel 4.8 - 4.10
Englisch
Das Maschinensprachebuch zum C64 & C128, 1. Auflage 1985, DATA BECKER

Kapitel 5.8 - 5.8.6
Gelfand, Felt, Strauch, Krsnik
Anti Cracker Buch, 2. Auflage 1988, DATA BECKER

Kapitel 9. - 9.3.4
Herrmann
Floppy 1541 - Pflegen und Reparieren, 1. Auflage 1985, DATA BECKER

Kapitel 10. - 10.13.4
Paulissen
Das Casettenbuch zu Commodore 64 und VC-20, 1. Auflage 1984, DATA BECKER

Anhang C.
Tornsdorf, Kerkloh
Das große GEOS-Buch, 3. Auflage 1988, DATA BECKER

Es kann sein, daß der eine oder andere der hier aufgeführten Titel vom DATA BECKER Verlag nicht mehr lieferbar ist. Im Fachhandel oder in gut sortierten Computershops können diese Titel jedoch noch vorrätig sein. In diesem Fall wenden Sie sich bitte mit o.g. Angaben an Ihren Fachbuch- oder Computerhändler.

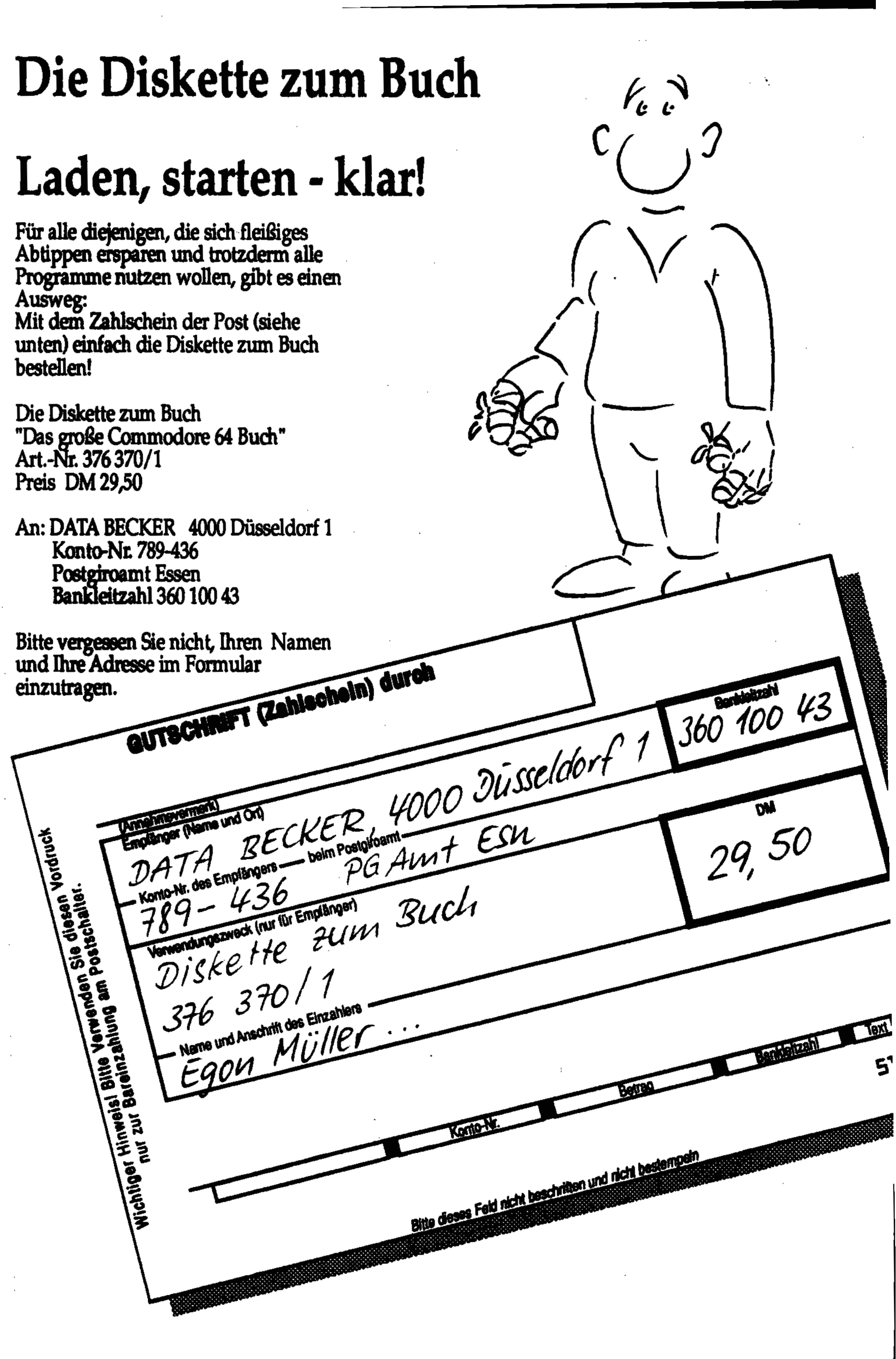
Stichwortverzeichnis

W

X

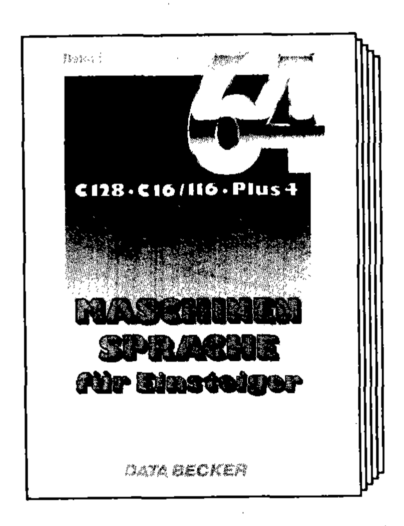
Y

Z


Die Diskette zum Buch
Laden, starten - klar!
Für alle diejenigen, die sich fleißiges Abtippen ersparen und trotzderm alle Programme nutzen wollen, gibt es einen Ausweg:
Mit dem Zahlschein der Post (siehe unten) einfach die Diskette zum Buch bestellen!
Die Diskette zum Buch
"Das große Commodore 64 Buch"
Art.-Nr. 376 370/1
Preis DM 29,50
An: DATA BECKER 4000 Düsseldorf 1
Konto-Nr. 789-436
Postgiroamt Essen
Bankleitzahl 360 100 43
Bitte vergessen Sie nicht, Ihren Namen und Ihre Adresse im Formular einzutragen.
GUTSCHRIFT (Zahlschein) durch
Bankleitzahl
360 100 43
(Annahmevermerk)
Empfänger (Name und Ort)
DATA BECKER, 4000 Düsseldorf 1
Konto-Nr. des Empfängers
beim Postgiroamt
789-436
PG Amt Esn
DM
29,50
Verwendungszweck (nur für Empfänger)
Diskette zum Buch
376 370/1
Name und Anschrift des Einzahlers
Egon Müller ...
Wichtiger Hinweis! Bitte Verwenden Sie diesen Vordruck nur zur Bareinzahlung am Postschalter.
Konto-Nr.
Betrag
Bankleitzahl
Text
Bitte dieses Feld nicht beschriften und nicht bestempeln

Maschinensprache verständlich für jedermann.

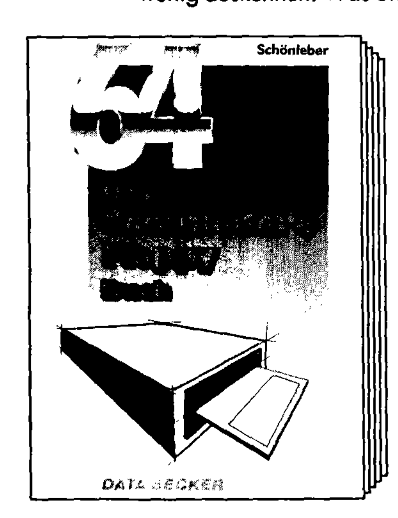
Endlich einmal kein unverständliches Lehrbuch, sondern ein Buch, mit dem wirklich jeder, der sich dafür interessiert, schnell und einfach Maschinensprache lernen kann. Ohne das übliche Fachchinesisch wissen Sie schon bald, was ein professionelles Programm ausmacht: BASIC-Routinen heranziehen, Befehle und Strukturen vergleichen und schließlich selbst in Assembler umsetzen – durch dieses Konzept sind Sie sehr schnell in der Lage, die Vorteile dieser Sprache für eigene Anwendungen voll zu nutzen. Eine echte Chance für alle Assembler-Interessierten, denen diese Sprache bisher zu schwierig erschien. „Sprechen" Sie die Sprache der Profis – mit Maschinensprache für Einsteiger. Geeignet für den C64, den C128 und auch für die ganz Kleinen (C16/C116/Plus4).



Baloui
Maschinensprache für Einsteiger
345 Seiten, DM 29,-
ISBN 3-89011-182-3

Der perfekte Einsatz Ihrer Floppy-Station.

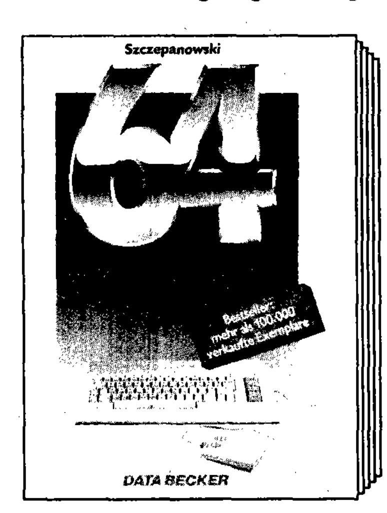
Mt der Floppy läßt sich weitaus mehr machen als nur laden und starten. Man muß sich lediglich ein wenig auskennen. Was Sie alles aus Ihrer Floppystation herausholen können, zeigt Ihnen das Commodore Floppybuch. Vom Einstieg bis zur Programmierung der Floppy in BASIC. Hier finden Sie alles über den Aufbau von Disketten, zu den einzelnen Dateitypen, zu den Systembefehlen und natürlich auch zu den verschiedenen Fehlermeldungen der unterschiedlichen Floppystationen. Was Sie über Ihre Floppy wissen sollten, finden Sie in diesem Buch – ob Einsteiger oder Profi und egal mit welcher Floppystation Sie arbeiten: der 1541, der II/41 oder der C/70/71/81.



Schönleber
Das Commodore-Floppybuch
240 Seiten, DM 29,-
ISBN 3-89011-269-2

Mit Schwung und Laune Neues lernen.

Anfangen und gleich loslegen – das wünscht sich jeder, der in die Computerei einsteigt. Mit „64 für Einsteiger" geht das ganz problemlos. Vom Anschluß bis zum ersten Programm. Systematisch und leichtverständlich lernen Sie hier Ihren neuen Rechner kennen. Mit zahlreichen Anwendungsbeispielen, Erklärungen zur hochauflösenden Grafik und natürlich mit einer detaillierten Einführung in GEOS 2.0 deutsch. Ein umfangreiches Lexikon macht das Buch auch dann noch wertvoll, wenn Sie längst kein Einsteiger mehr sind. Aufstellen, anschließen, kennenlernen, das erste Programm entwickeln – Sie werden Spaß daran finden, etwas Neues zu lernen. 64 für Einsteiger – und Sie sind bestens für eine eigenständige Arbeit am Rechner vorbereitet.



Szczepanowski
64 für Einsteiger
251 Seiten, DM 29,-
ISBN 3-89011-010-X

Das Nachschlagewerk für zukünftige GEOS-Profis.

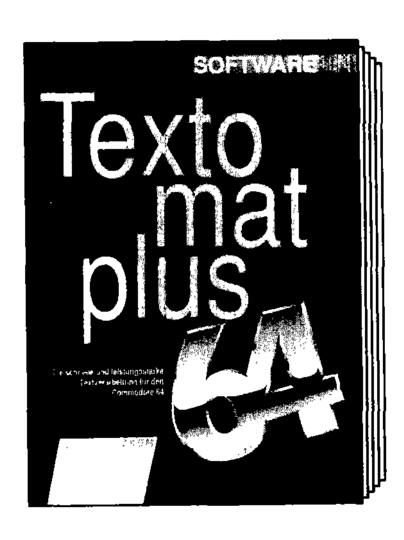
Das große GEOS-Buch – unentbehrliche Grundlage für alle zukünftigen GEOS-Profis. Hier finden Sie alles, was es zu GEOS zu sagen gibt: Wie ist das GEOS-File-Format aufgebaut? Wie erstelle ich eine bootfähige Sicherheitskopie? Wie schreibt man Programme mit GEOS-Eigenschaften? Wie bekomme ich einen Maschinensprache-Monitor in GEOS? Sonst noch Fragen? Einfach nachschlagen. Das große GEOS-Buch wird Ihnen alle Fragen zu GEOS Schritt für Schritt beantworten. Natürlich mit jeder Menge Tips & Tricks für Ihre tägliche Arbeit. Bei alledem werden selbstverständlich die GEOS-Version 1.2 und 1.3 besonders intensiv behandelt. Das große GEOS-Buch – geeignet für den C64 und C128.



Kerkloh/Tornsdorf
Das große GEOS-Buch
424 Seiten, DM 49,-
ISBN 3-89011-208-0

Einfach & schnell: TEXTOMAT PLUS 64.

Von einfachen Blockoperationen bis zum Erstellen von Serienbriefe – z.B. mit DATAMAT, bietet Ihnen TEXTOMAT PLUS 64 alle Vorteile einer schnellen, leistungsfähigen Textverarbeitung. Selbst die Kombination von Text und Grafik beherrscht dieses Programm perfekt. TEXTOMAT PLUS 64 – die Textverarbeitung mit dem Leistungsplus: Komfortables Suchen und Ersetzen, komplette Bausteinverarbeitung, beliebig lange Texte durch Verknüpfung, frei programmierbare Steuerzeichen, Serienbrieferstellung, Floskeltasten, Wordwrap, frei einstellbare Tabulatoren, Rechenfunktionen für alle Grundrechenarten, Mischen von Grafiken und Texten u.v.a.m. TEXTOMAT PLUS 64 – die ideale Textverarbeitung für Ihren 64er. Zu einem unverschämt günstigen Preis.



TEXTOMAT PLUS 64
unverbindliche Preisempfehlung DM 59,-

Das große Commodore 64 Buch

Der C 64 ist ein attraktiver und preiswerter Computer, der vor allem Hobbyanwendern und Programmierern entgegenkommt. Wer diesen Computer nicht nur zum Spielen verwenden will, sondern auch die phantastischen Grafik- und Soundmöglichkeiten voll ausnutzen möchte, dem zeigt das vorliegende Buch eine komplette Darstellung aller Fragen, die in diesem Zusammenhang auftreten können:
Von der BASIC-Programmierung über die Floppyansteuerung bis zur Sound- und Musiktechnik werden neben vielen praktischen Programmen Tips und Tricks vermittelt. In den Anhängen finden Sie schließlich alles, was man beim Umgang mit dem C 64 braucht: Befehlsreferenzen, Zeropage, Fehlermeldungen, diverse Systemtabellen u.v.m.

- Die Textverarbeitungs-, Grafik-, Desktop Publishing- und DFÜ-Möglichkeiten des C 64
- Mit der Benutzeroberfläche GEOS 2.0 arbeiten
- Das A und O der Dateiverwaltung
- Den 6510-Mikroprozessor in Assembler programmieren
- Sprites perfekt entwerfen und mit ihnen umgehen
- Den Sound-Chip des C 64 programmieren
- Die Wunderwelt der hochauflösenden Grafik kennenlernen
- Die Schnittstellen des C 64 optimal einsetzen
- Kleinere Reparaturen der Floppy 1541 selbst durchführen
- Die Datasette 1530 nutzen und mit ihr umgehen.

Wenn Sie ihren C 64 optimal einsetzen, in BASIC und Assembler programmieren oder es lernen wollen, dann benötigen Sie dieses Buch.

ISB N 3-89011-370-2 DM +029.80

DM 29,80
ÖS 232,–
sFr 29,–

DATA BECKER

